AD-A100 784 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
A FUNCTIONAL LEVEL PREPROCESSOR FOR COMPUTER AIDED DIGITAL DESI--ETC(U)
DEC 80 P G RAETH
UNCLASSIFIED AFIT/GCS/EE/80D-12 NL

1 of 3
AD
A100784

DDC

LEVEL II

AD A100784

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

81 6 30 039

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

AFIT/GCS/EE/80D-12

A FUNCTIONAL LEVEL PREPROCESSOR FOR
COMPUTER AIDED DIGITAL DESIGN

THESIS

AFIT/GCS/EE/80D-12 PETER C. KAETH
2LT USAF

AFIT/GCS/EE/80D-12

A FUNCTIONAL LEVEL PREPROCESSOR FOR
COMPUTER AIDED DIGITAL DESIGN.

THESIS

PRESENTED TO THE FACULTY OF THE SCHOOL OF ENGINEERING

OF THE AIR FORCE INSTITUTE OF TECHNOLOGY

AIR UNIVERSITY

IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

BY

PETER G. RAETH ASEET, BSEE

2LT USAF

GRADUATE COMPUTER SCIENCE

DEC 80

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

This work is dedicated to my parents,
two God fearing people who passed the
RAETH heritage on to their children.

Come, let us sing joyfully to the
   Lord;
Let us proclaim the Rock of our
  salvation.

PSALM 95:1

TABLE OF CONTENTS

LIST OF FIGURES

PREFACE

There is no such thing as a one man show in the engineering profession. Verily, many people have given me their willing and able support during this project. Principle among these has been my thesis committee. Dr. Gary Lamont as chairman provided much guidance as this my most ambitious project began. Being a software oriented person by trade and hobby, I had never gotten much involved with hardware at the chip level. Dr. John Borky as a committee member provided me with several important insights into the performance of various chips of the MOS variety.
Dr. Walter Seward, the other committee member, helped me to get started on the DEC System 10, the primary tool of this investigation.
All three proved quite patient with my naive beginnings and later with my attempts at writing.

Money and resources must come from somewhere and these were eagerly provided by the two sponsors. Mr. John M. Acken; Sandia National Labs; Dept 2113; Albuquerque, NM, 87185 gave me much of his time and a gate level digital systems simulator which he maintains. He also welcomed me to his home and office during my TDY to Sandia Labs. His suggestions as to what needed to be accomplished provided the initial framework for the project.

Capt. John B. Rawlings; AFWAL/AADE-3; WPAFB, OH, 45433 was the other sponsor. He and his colleagues: Mr. Mike Mills, Lt Eric Smith, Mr. Rick Stormont, Lt. Joe Tatman, and Lt. Mike Tebo gave constant feedback on the real world requirements of Computer-Aided-Design. They were always ready

with their time and fellowship. Capt. Rawlings' initiatives at AFIT opened the door for this investigation. His division, headed by by LTC. Gary Pritchard, saw to the availability of the DEC System 10 and a well equiped office.

The library services of AFIT were superb. Mrs Molly Bustard always assisted as her shelves were gradually emptied. She helped in finding books and in long term withdrawals. Mr. Stan Boyd gave his excellent aid to the filling of requests for quite a number of backdated journal articles. What our library did not have, he searched the world for.

During their inception, many projects benefit from the counseling of people who have a good feel for what will be acceptable and what will not. In this regard my thanks go out to Dr. Tom Hartrum, Dr. Kenneth Melendez, and Dr. Jim Rutledge.

Not to go unmentioned are two very fine technicians in AFIT's labs. Mr. Dick Wager and Mr. Dan Zambon saw to it that books and equipment held by their department were made available. They also gave informative discussions on the acutal use of MOS chips.

Also there is Mr. Mike Culp, a high school student who studied computer programming under my tuteladge. He produced several of the descriptive drawings that were needed.

Finally, but certainlly not in the least is my dear friend and colleague Capt. Nadine Levine. Her intuition helped me to solve the several problems which bedevil thesis students. Her never ending encouragement made doing this thesis much easier.

To all of these people I owe my sincere gratitude.

Peter G Roth

ABSTRACT

While good gate level and register transfer level digital simulators exist, one can not easily integrate the two due to their inherent limitations. A given simulation can not be described partially in gate level and partially in a higher level. A solution is to create a functional level preprocessor and a library of functional device models linked to a gate level simulator's input language. This permits the mixing of behavioral models with gate level models in the same system structure. The combination of processes (element models or primitives) and their structure (interconnections) can be exercised all at one time during a single simulation session. From the start, there came forth an obvious method which could be used to intermix the several levels of modeling (*).

Two separate pieces of software were written to implement a specific solution to the above stated situation. SISL, Structural Interface to the Salogs Language was created. This is a functional level preprocesor to SALOGS (SAndia LOGic Simulator) which is an eight-state, MOS, gate level digital systems simulator. SISL will accept functional level systems descriptions and convert them to a form acceptable to SALOGS.

The other effort was the building of a functional level modeling library. This library consists of three behavior models: a 4-16 decoder, a 2048 X 8 ROM, and a 256 X 8 RAM. These models are designed to be used in a functional level/gate level model of a digital system and will link to the SALOGS run time system. Together, these two programs (SISL and the modeling library) provide the easy use of the top-down approach to digital system design. Thus, the project's culmination.

(*) See Chapters 3 and 7 for more on the hierarchy of digital systems modeling.

The result of this investigation was the new ability to easily mix functional and gate level models during the same simulation run. A system may be described both in functional and gate level primitives. This is necessary because the USAF is constantly increasing its use of very large scale integrated circuits (VLSI). It is uneconomical to simulate systems which use these circuits at the gate or register transfer levels due to the computer and human resources required.

CH 1. INTRODUCTION

Gate level simulation along with register transfer level descriptions has been the bread and butter of computer aided logic design [H2,86]. Until the present time, eight-state, gate level simulation of digital devices has sufficed for the development of most logic circuits. The eight states are: low level, undefined, high impedence, high level, negative slope, transient undefined, transition to high impedence, and positive slope [A1,12].

In the past, many simulator packages have been restricted to gate level models because the state of the art would not support more general types of modeling [S9,25]. This restriction has caused many problems as the state of the art in digital device construction has improved.

PROBLEM

One of the major problems of digital simulation is that large systems are prohibitively difficult to describe and simulate at the gate or register difficult to describe and simulate at the gate or register transfer levels. transfer levels. The desire, therefore, was to create a new level of simulation called the functional level [H2,86].

At this level the designer can specify subsystems such as ROM, RAM, busses, shift registers; in short, logical devices of arbitrary complexity. Functional level modeling, as addressed by this investigation, is meant occurs during the "...circuit description language..." phase of the typical CAD (Computer Aided Design) run.

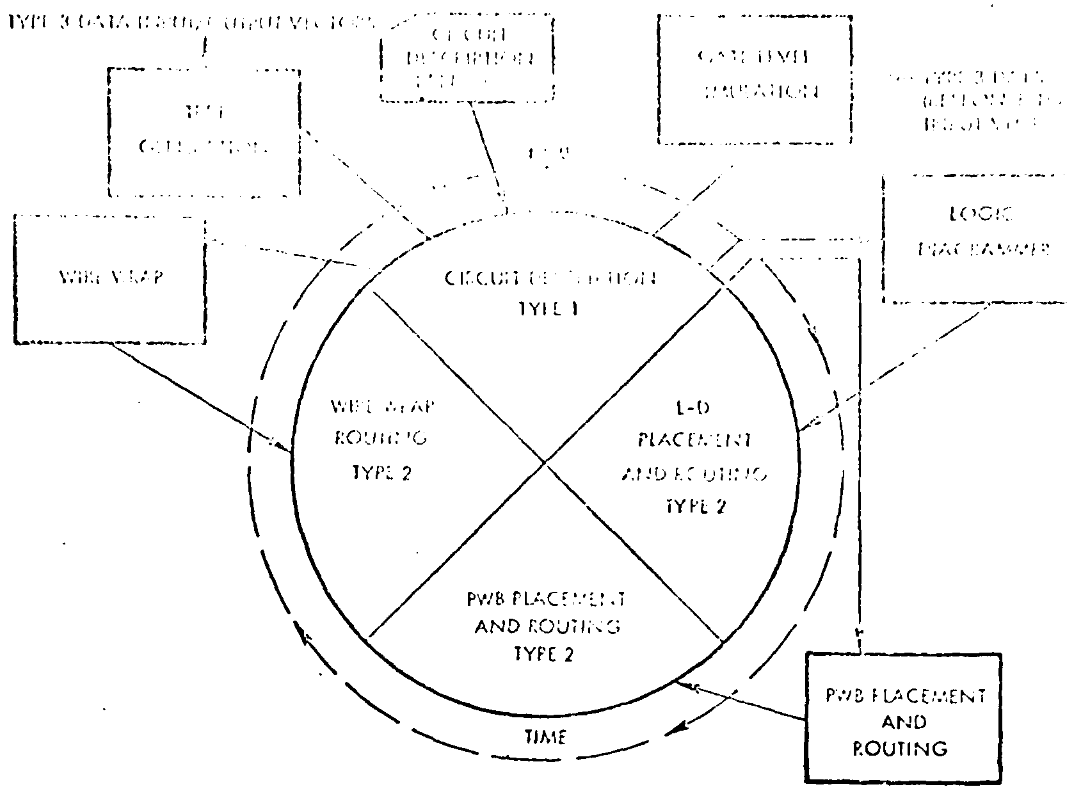


FIG CH1-1

A TYPICAL CAD RUN (C8,292)

The modeling and simulation of a digital system follows a pattern which enhances the use of machine assistance. Material developed via simulation can be used in the actual implementation as well. Fig CH1-1 [C8,292] depicts the simulation proces-. The intent is to affect only the indicated phase and to let the rest of the simulation to proceed as if no modifications had taken place. At this phase functional/gate level models are described, not just gate level models as in the past. SISL does its work at this node and SALOGS picks up the process at the "...gate level simulation..." node.

GOAL

It should be possible to allow computer designers to focus on certain portions of a digital system's logic while ignoring details of other portions. Designers should be able to "black box" certain portions of a big system. At times it will not be known what the future contents of a module will be, only that it will have to exist in the system. The results of this study should allow the designer to choose which blocks to describe at the gate level and which to describe at the functional level. A person would thus be encouraged to follow a top-down design methodology. A preprocessor to the gate level simulator should handle the connections between the two levels of descriptions.

It is possible to develope software which can give a great deal of help to the designer during any attempts at higher level models and simulations.

APPROACH

The above goal was met by designing a library of functional level subroutines. A functional level preprocessor was also developed. The preprocessor accepts digital descriptor input and converts it to a form which logically links the subroutine library to the input language of a commonly used gate level digital simulator. Such a common industrial simulator is found in SALOGS, an eight-state computer-aided-design (CAD) system developed at SANDIA LABS [C2,1] [Appendix D].

Salogs allows the user to describe models composed of MOS gate level primitives (AND, OR, etc.) and to perform simulations using those models. It will also do fault analysis. Another feature is a capability to accept subroutines which are callable as modeling primitives.

This approach to the realization of the goal must necessarily be clearly defined as to what may be expected of it.

SCOPE

The preprocessor mentioned above does not convert functional descriptions into gate descriptions. It does, however, create the logical linkages to the SALOGS input language so that pin connections, device names, and other parameters required by SALOGS are made consistent with the users' gate level portion of the overall system being modeled. The preprocessor delivers two outputs based on the functional level description it receives: functional identifiers required by SALOGS and a model specifying the overall functional system. From these outputs, the preprocessor creates a file of functional descriptors which can be appended to the gate level portion of the users' description.

The user must identify his interconnections to the preprocessor's circuit. Since SALOGS allows the linking of subroutines to its own code, no modifications have to be made to SALOGS itself. The software is run in batch mode due to the extended amount of wall time and core required by the SALOGS software (*). To ensure portability, the project was done in ANSI 66 standard FORTRAN IV, the same as SALOGS itself.

The library of functional models is written to deliver useful information to the designer. These models (which account for all eight states), under specific inputs will indicate that an unspecified input has occured rather than behave as the real circuit would.

There are some o Ther ideas and features which should be presented. These have to do with the development of the software and techniques of functional level modeling.

(*) Wall time varies according to how many jobs are currently being handled by the computer system. Core is constant at around 200K for each program in the SALOGS/SISL series.

FOR FURTHER DISCUSSION

The body of this thesis is concerned with three ideas: digital modeling in general, the methods of functional modeling, and the internal workings of the SISL preprocessor.

Chapter 2 gives an introduction to some of the basic ideas used by those who actively engage in the simulation of digital systems. Some approaches to the subject are generally accepted in the simulation community as standard and this chapter reviews these. Chapters 3 and 4 discuss the top-down method of digital system design and its specific application to this project.

Chapters 5 and 6 present some general methods of creating behavioral models of digital devices. These were developed for use in this investigation and are applied to the modeling library.

In Chapters 7 and 8 one will find an overview of digital system representation levels and a few of the languages used to work at one or more of those levels. SISL is an application of the ideas found in these other digital system description languages.

The thesis closes with the summary and conclusions found in Chapter 9.

These presentations are supported by an Appendix and Glossary. In the Appendix will be found users' guides to the various software, listings of the SISL and modeling programs, sample runs, and flowcharts. The Glossary defines the specific terms used and will clear up any confusion as to their meaning.

CH 2. AN OVERVIEW OF THE COMPUTER AIDED DESIGN OF DIGITAL SYSTEMS

This chapter reviews some of the basic approaches to digital systems simulation. It begins with a definition of what computer aided design should accomplish and goes on to the general ways in which one may employ simulation techniques for digital systems. A summary provides information on the current uses of simulation.

DISCUSSION

Because of the complexity and economics of today's digital systems, a design tool is needed that will allow the error free analysis and testing of circuit implementations (*). This tool should not require the physical realization of the circuit. Such a tool has been found in the form of a computer running a piece of software which will exercise a digital system that has been described in the input to that software. Programs of that nature performing computer-aided-design (CAD) permits the digital designer to submit his circuit ideas to strict and nearly complete simulation and analysis without having to physically construct the hardware.

Many such simulation tools exist today [B2,1] [C2,1] [V1,1] [V3,1]. (See the Bibliography for papers on specific languages.) They provide an entrance to the solution of modeling problems. Most of the larger companies in the computer industry are involved in CAD research. Among these is IBM [B1,20].

(*) Unless otherwise noted, the references for this chapter are found in the works by Acken and Case listed in the bibliography.

The simulation of a digital system is the description of the system model in an appropriate computer language along with the computer's experimentation with that model. Through the use of CAD software, a circuit description and operating parameters are taken as input, with experimental and statistical results of the circuit simulation as output.

TWO GENERAL TYPES OF SIMULATION

There are two general types of logic simulation: True-Value Analysis and Fault Simulation. A designer using True-Value Analysis is judging the circuit's ability to perform according to the original specification of the design criteria. Fault Simulation is employed to observe system operation under various forms of circuit flaws.

(TRUE-VALUE ANALYSIS) In its most fundamental form, True-Value Analysis is the acceptance of the logical description of a circuit, the application of logical values (1's or 0's) to the circuit's inputs, and delivering as output the Boolean result of the combination of circuit and inputs. An extension of simple Boolean modeling in terms of the binary values 1,0 is to add a state to the simulation called a DON'T KNOW or UNDEFINED (or *). This unknown logic state is distinct from a DON'T CARE whose logic level can be either 1 or 0 with no affect on the operation of the circuit.

Given smaller and smaller time steps, the time it takes a voltage to rise to the 1 level or fall to the 0 level becomes important. As digital circuits become faster and faster, the timing issue becomes more important. Thus, more advanced simulators use three extra states:
D, negative slope; U, positive slope; and X, transition undefined.

Six-state simulation allows the computer modeling of very fast digital systems without worrying about the particular technology to be used in the actual system construction. Some simulators (SALOGS for one) do incorporate MOS technology in their software. These simulators employ two additional states result in an eight-state simulation. Those two states: H, high impedance; and A, transition to high impedance are used to simulate a device being effectively off-line.

Digital simulators which model various other technologies usually have the ability to be set for four- or eight-state mode. In four-state mode, the model operates using, high, low, undefined, and high impedence. Eight-state mode simulates using the added states of negative slope, positive slope, transition undefined, and transition to high impedence (See Appendix D).

(FAULT SIMULATION) Fault simulation is performed to derive a set of input signals which can then be used as stimuli to test the functioning of a logic network. These signals form a test pattern which can be auutomatically generated by the simulation software. When these signals are placed on a circuits' input, they allow the detection of certain defects [T3,38]. Usually, single "stuck-at" fault modeling is used due to the difficulties of multi-fault modeling. In this method, only one defect is assumed and it is a particular signal being "stuck at" or a "never changing" value. Either one of the module's outputs is stuck or one of its inputs is stuck. There are four ways to simulate a fault: fail-all simulation, which will fail all outputs one at a time; parallel fault simulation, which simulates several faults at once; deductive fault simulation, which lists faults which cause a change in the output of a given module compared to the unfaulted circuit; and concurrent faulting, which only simulates the parts of the faulted circuit whose inputs, outputs, or states do not agree with the unfaulted circuit.

SUMMARY

Since it has become so expensive to build and test unproven hardware, computer simulation of digital circuits is being employed more often by the military and industry. Since design correctness can be verified without actual hardware realization, the cost of design and implementation is cheaper than it would be if computer simulation were not used. Simulators make it possible, without risk to a physical circuit, to study and experiment with a system or subsystem. (There is, however, a certain financial risk associated with committing a facilities' resources to the simulation task [S9,23].) Simulators make fine pedagogical devices for teaching both students and practitioners the variations of the design and analysis of digital systems. Perhaps one of the most important benefits from an engineering point of view is that they allow systems to be exercised under expanded, compressed or normal timing. Overall, they permit the designer to judge his designs conceptually without actually having to build them.

Work on such design aids has been pursued by Sandia Laboratories (Albuquerque NM), the Avionics Laboratory of the Wright Aeronautical Labs and A r Force Institute of Technology (Dayton, OH), to name a few. As more and more standard models of low level devices are created, digital modeling at higher and higher levels becomes possible, thus overcoming the bottleneck of man-years and computer resources required to create simulations of large scale digital systems.

CH 3. THE BUILDING OF A SYSTEM USING FUNCTIONAL MODELS

A digital system is not simply created in its final form. It is not usually possible to design a working product on the first attempt. Several systems are developed in the course of a development effort. These range from the interconnection of a few high level subsystem blocks to the detail of a gate level or lower model. This chapter introduces the reader to the process of going from the higher, undetailed modeling level to the lower, detailed level.

THE GATE LEVEL AND BLACK BOX BEGINNING

The desire to intermix gate and functional models derives from the hierarchy of a top-down approach to digital systems simulation. In this method, one begins with an undetailed viewpoint of the desired system. This viewpoint is in terms of a few general blocks. As the modeling effort goes on, these blocks are broken down into more detailed sub-blocks. Finally, each sub-block is defined by progressively more complex units until the desired level of detail is achieved.

Thus, the black box is a part of a model which, as yet, does not perform as it ultimately will. It can be connected to more detailed portions such as a block described in gate level detail. There is a certain technique to using this mix when simulating with SALOGS.

USING THE BLACK BOX

SALOGS has the capability to "SET" the value of any of its nodes. (For more discussion on SALOGS see the SALOGS USERS GUIDE in the Appendix.) Such nodes will retain their set value regardless of system operation. Therefore, the designer can allow the black box to either deliver some default output for any input or deliver a SALOGS set output. The model may then be studied under various conditions which may be eventually produced by the future contents of the box. The default outputs can be used to flag the fact that the box would have had some effect on the systems' operation.

EXPANDING THE BLACK BOX

Gradually, decisions will be made as to the required output of the box given certain inputs. Now the functional model may be expanded to produce that output when the stated inputs occur. A default to some flagging output (such as undefined) can be arranged for non-specified inputs.

As more data is gathered and greater detail is developed, the black box becomes a true functional model performing very nearly as its gate level counterpart would. It has the advantage of using less core and time to run and it ignores some of the unwanted or unneeded detail attendant to gate level models. It can be expanded to any desired level of detail depending on resources and the needs of a given simulation.

CH 4. SIMULATING BI-DIRECTIONAL LINES

SALOGS, the gate level modeling software, has some particular requirements when bi-directional lines are called for. This chapter introduces the background s of such lines and continues with the detailss of their simulation in SALOGS.

INTRODUCING BI-DIRECTIONAL LINES

As chip manufacturers have heaped complexity upon complexity, the number of pins necessary for I/O has increased. Fourty pins has generally become the acceptable maximum standard but some chips would require more than that. Because of this, certain pins have been designed to carry data in both directions. These pins thus make it possible to have fewer connections to a chip while keeping the original number of options.

The issue of bi-directional lines should be addressed. If a design aid is to maintain its capability to deal with state of the art systems, it must accomodate the devices which make up those systems.

SALOGS SIMULATION OF BI-DIRECTIONAL LINES

SALOGS itself will not allow the direct use of bi-directional lines. Nodes are either input or output but not both.

Without modification to the SALOGS package, it is not possible to truely simulate bi-directional lines. However, by using a buss model and splitting each two-way line into one input and one output line, one can still model devices with such lines. Along with this, one can incorporate timing and clocking and delay parameters in the buss. (The behavioral models themselves do not contain these parameters.) Such a splitting out of bi-directional lines has not proven to be a drawback to the modeling effort of this investigation. The RAM and ROM models to be discussed later use line splitting.

The buss model also solves a problem caused by the way SALOGS updates nodes. Nodes are updated sequentially, one at a time each time step. When bi-directional lines are used the wrong item could be updated first. Let us say, for instance, that the CPU is talking to the RAM. If the RAM is updated first then the CPUs' input is not properly considered. The buss is last to be updated so that the RAM and CPU get correctly updated in the next time step. SALOGS is caused to update the buss last when the user lists the buss last in a system description.

The buss model could be expanded to also handle buss contention. While SALOGS can easily handle the fanout of output lines, it can not describe the fanin of input lines. Only one output node can talk to any given input node at any one time. If input to a given node can come from more than one output node, some buss control must take place.

A SALOGS bi-directional buss can be written as a behavioral model to handle several simulation requirements. The most important of these are two-way lines. These are followed by timing, clocking, and delay parameters. One final item is the description of a buss contention controller.

CH 5. CREATING A CHIPS' BEHAVIORAL MODEL

Several issues must be considered when writing software which describes a behavioral model. It is not a straightforward task to construct such a program. Presented in this chapter are the methods used and the considerations taken in creating the behavioral modeling library.

COMPLEXITY VS. DETAIL

Many are the ways to create a behavioral model. Each method has its own advantages and drawbacks.

Let D= detail of simulation
C= device compexity
K= a constant representing computer and human resources

Then D * C = K would be an excellent conceptual formula for describing the various limitations to face when modeling a digital system (*).
The overall goal of modeling is to simulate in great detail very complex devices while minimizing core and human involvement.
These ideas are very much at odds with each other. As the complexity of the device increases, the limitations of computer and human resources prevent the simulation of a great amount of detail. Conversely, if a large amount of detail is required the same problem will not allow the modeling of complex devices. As the available human and computer resources increases or decreases so can detail and complexity to a proportional degree.

The following sections review several methods of creating behavioral models. These were developed in the course of the attempts made to create an economical yet detailed behavioral modeling library. There will be an [illegible] tradeoff evident in that, while a particular method may be easy to implement, it may not always yield an economical or otherwise usable model.

(*) This formulation was originally suggested by the sponsor, Rawlings.

SIMPLE TABLE DRIVEN MODELS

The fastest way to derive an output from an input is to map the input to a location in a table which contains the corresponding output. It is helpful to assign a number to each of the eight states that any given node may attain:

SALOGS Assignment	FORTRAN Assignment	State
0	1	0 False
1	2	* Undefined
2	3	H High Impedance
3	4	1 True
4	5	D Downward Slope
5	6	X Transient Undefined
6	7	A Transition to High Impedance
7	8	U Upward Slope

When SALOGS fixes a node value it uses these assignments. These must be converted to the FORTRAN assignments for array table access. As an illustration, consider a box which has two inputs and two outputs. The inputs can be modeled using a two dimensional table which is 8 X 8. The output lines are modeled the same way only with an 8**2 X 2 table. There are 8**2 output possibilities due to the 8 X 8 possible input arrangements. The following is an example of how the array tables are used to model the box. Let input-line-1 be in state A and input-line-2 be in state U. This will cause a certain resulting output. SALOGS will represent the input event as 6,7. The input table will be accessed using (6+1,7+1). Contained in that location is the row of the output table which holds

the required output line values. Each column of the output array holds a state for a designated output line. In general terms, an 8 X 8 x...x 8 input table maps to an 8**N X K output table where:

N= # input lines
K= # output lines

If K=1, then the value found in the input table is the state of the output line. States assigned to output lines are SALOGS assignments.

In this case, due to the stated behavior of the box, there may not be 8**2 unique output arrangements. If that proves true, the output table may be shrunk accordingly to an M X 2 array; where M is the number of unique outputs. M's maximum value is 8**2. The input table remains the same size, but any given location could hold the same value as another. In general, there is a maximum of 8**Z unique outputs; where Z is the number of output lines. Also, it may not matter which is input-line-1 and which is input-line-2. In other words, (input-line-1 +1,input-line-2 +1) may always yield the same value as (input-line-2 +1,input-line-1 +1). In that case only the upper or lower triangle of the input matrix would be needed. In FORTRAN, though, it is not possible to dimension a triangular array. If carefully documented, the unused portion of the input table could be applied to some other activity, thereby achieving a savings in core.

Once the output array has been accessed, one needs to find there the values which correctly define the devices behavior.

(DERIVING THE OUTPUTS) The next question involves exactly what outputs are required from all the possible input combinations. For an original circuit, a knowledge of the chips' technology and configuration would be the key. For pre-manufactured circuits, there is available an industrially proven and tested gate level modeling package, SALOGS. Its primitives (AND, OR, NOR, etc.) are fully defined MOS models. They will deliver a correct output given any combination of the states as input. With this package one could model a device at the gate level, apply all possible input combinations, and thus receive a corresponding list of outputs. This list can then be used to load the output table. The gate model need be run only once. Its results can be held in off-line storage until the data is needed to load the I/O arrays.

A problem with the above technique is that a gate level model taken from a data book is only a logical model, not an operational one. Also, it would be extremely difficult to model, say, a 64K RAM at the gate level. So there are limitations to just how far one can go with this method. Too, as the chip becomes more complex, a lot of core is required to represent the I/O tables. (A five input OR gate requires 8**5 array locations.) On the other hand, not much time is used in the simple array mapping process. These problems can be, in part, overcome by the following technique.

TABLE/EQUATION DRIVEN MODELS

If a certain state on any input line always causes the same output arrangement, the I/O tables can be collapsed accordingly. Equations, both arithmetic and logical, would be then required to recognize the states which cause fixed outputs. Other arguments would be needed to map the other input combinations to the reduced tables. For example; consider the five input OR gate which used 8**5 array locations. Employing a combination of equations and tables this can be reduced to 7**5. A true on any input line causes a true output in an OR gate. Similar thoughts affect the multiple input AND gate. Any input being false will cause a false output.

Depending on how many states cause constant outputs, this method may use less core than the previous one. However, the designer must deal with the state recognition and mapping equations. These logical/arithmetic computations may consume more time and core than the simple direct mapping arguments. Experience has shown, however, that this method presents important advantages over the simple table driven models. It is possible to carry the use of equations even further as the next method will demonstrate.

TRUTH TABLE/LOGICAL EQUATION DRIVEN MODELS

Devices of lesser complexity than, say, a CPU are described in the data books by a truth table. Since logical AND and OR functions are a part of the ANSI standard FORTRAN, it is possible to write logical expressions to represent output vs. input. These equations take the form ABC+(-A)BC=0 and so on; where the left hand side is input and the right output. Each input value is stored in its binary form (3=011 for instance.) A logical manipulation of the bits representing the input values can give the output values.

By using logical equations, output values can be derived from input values such that the output values are those indicated by the devices' specification sheet. These logical arguments can also be implemented using AND/OR eight-state mapping tables. For instance: the 4-16 decoder to be described later has 16 equations each of which access tables representing a four input OR gate and a one input inverter.

Extensions to the aforementioned equations will have to be created if the specification sheet only specifies certain input events. Some data sheets do not specify the results of all eight states on input, only the results of true and false. Others specify rising and falling slopes. It is still necessary for the designer to account for all eight states when creating a behavioral model since SALOGS could place these states on the input lines. Some designers simply make the outputs all undefined if any but the truth table values appear on the input. This is acceptable as long as the overall modeling goal is reached. The desire here is usually to perform logic verification. The valid inputs (in the sense of the allowed input space) could be expanded to cover any of the eight SALOGS states.

The tradeoff in core and time must be carefully considered here. A large program can take up as much core as a sizable array and run much slower than a simple table driven model. Care should be taken to simplify as far as possible not only the code but the logical equations as well. The tables which represent the eight state OR and AND gates can be considered as free because they are accessible by more than one model.

The last method is used to model complicated VLSI circuits.

FUNCTIONAL DESCRIPTION MODELS

For the most complex of devices, the data books provide a functional description of the chips' operation. A FORTRAN program can then be written to describe this functional behavior. Certain results will be guaranteed by the manufacturer. These will predict the output only for certain constrained inputs. Other input events must be accounted for by the model designer. That person must decide what should happen when events outside the manufacturers' specification occur.

MODELING FOR TRUE CIRCUIT OPERATION VS. SIMULATION RESULTS

Primarily, the designer is interested in knowing whether an unspecified event has occured. This is opposed to being concerned with what the chip will actually do under that stimuli. The prefered simulation result, in general, is an undefined output given unspecified inputs.

What a chip will do outside the valid event space depends on several factors. Among these are: chip technology (MOS, TTL, etc.), configuration, and a statistical model which represents which batch the individual chip came from. By and large, designers desire a model that works the same way all the time.

A device's configuration is usually proprietary information. Therefore, it is not always possible to know how the chip is put together and thus gain an idea of what will happen given all inputs. The manufacturer only guarantees and specifies the results given certain inputs. On top of this, designers do not always want what could be a recognizable output to result from an input which should not occur. They prefer some flagging output to mark the unexpected event. This is valid when simulating for logic verification and proper system performance.

CONCLUSIONS

There are several ways to model a chip. This discussion has covered simple table driven models, table/equation driven models, truth table/logical equation models, and functional operation models. A technique should be chosen based on what information is available on the device, resource limitations, and the detail required.

These methods were used in this investigation to derive behavioral models for three digital subsystems. A combination of methods was found to be helpful in realizing additional savings in the amount of computer resources required to implement a particular device's model.

CH 6. A FUNCTIONAL LEVEL MODELING LIBRARY

To fulfill the idea of functional level modeling advanced by this thesis, a library of FORTRAN models was created. A 4-16 decoder, 2048 X 8 ROM, and a 256 X 8 RAM were modeled and made to interface to SALOGS. They are supported by eight state models of an inverter and a four input OR gate. These were chosen because of the immediate needs of the sponsors. These needs reflect current projects which they have undertaken. A balance was struck between core and time usage. Each model reflects trade-offs in detail, complexity, and resources as well as in the accounting for the results of input events not mentioned in the data books. All of the models were tested by writing SALOGS routines which would exercise them through the several functional operations specified by the manufacturer. The tests were then compared with the expected results. In all cases, the simulations were found to perform as described in the data books.

The remainder of this chapter will be concerned with the three models, including their construction, operation, and use. Reference will be made to their block diagrams, flowcharts, and listings.

THE 4-16 DECODER

The 4-16 decoder is discussed first because it took by far the longest time to model. Techniques had to be developed to create various kinds of models and an understanding of the realities of chip use had to be reached.

A 4-16 decoder basically performs this function: The binary value on the four input lines is read and evaluated. Depending on that binary value, one of the sixteen output lines is set low; the rest are set high. For instance, 0001 on the input would couse output line 1 to go low and lines 0 and 2-15 to go high.

As a first step in modeling this device, a large photograph was taken of its gate level diagram found in the data book [N1,1-56]. Names were then written on each node. A SALOGS gate level model was next constructed to exactly represent the photograph. This model was then tested for results, outputs vs. inputs, to derive the eight state results of all the possible input combinations.

At first a simple table driven model was attempted. Required for this was an input array 8 X 8 X 8 X 8 and an output array 8**4 X 16. It was subsequently decided that this was a bit too much core even though the simulation would execute very fast. So a truth table/logic equation driven model was tried next.

To support this a table/equation driven model of a four input OR gate was created along with a simple table driven model of an inverter. The truth table of the decoder [N1,1-58] was then implemented by a series of 16 OR gates. These gates perform as would the gate level SALOGS four input OR gate. (SALOGS models a four input OR gate using three, two input OR gates.) The equation for each decoder output line is:

D+C+B+A ; D+C+B+(-A) ; D+C+(-B)+A ; ... ; (-D)+(-C)+(-B)+(-A).

Each decoder output line is driven by its own OR gate.

The following SALOGS code will allow the user to access the decoder:

```
$MODELS
ORDECOD 0 16 4 20 8 0
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15     *
A0 B0 C0 D0
END ORDECOD
$END MODELS
INPUT A0 B0 C0 D0
OUTPUT 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
ORDECOD 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 *
        A0 B0 C0 D0
END
```

If using the SISL preprocessor, the user would specify:

```
ORDECODE 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 ; A0 B0 C0 D0
```

and leave off the SALOGS $MODELS portion.

Appendix E shows the listing of the decoder modeling software. Refer to the Appendix and the SISL USERS GUIDE for more on specifying models to the preprocessor.

THE 2K X 8 ROM

A ROM "Read Only Memory" is a device which has a series of binary words pre-loaded into its memory. On demand, it will place the addressed word on its output lines.

Hardest to simulate were the unspecified input events. A functional operation specification [12,6-34] provided the basis for the final creation. Sandia Labs made decisions based on their viewing of proprietary information as to how the device should react if unexpected inputs occured. Appendix G shows the original block diagram of the ROM, the implemented block diagram, and the operational flowchart of the model. Appendix G also shows the listing of the software.

To access the model use the following SALOGS code:

```
$MODELS
ROM8  0 9 18 27 10 0
READY DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 *
CLK CE CEINV ALE RDINV IOMINV IORINV                  *
ADDR10 ADDR9 ADDR8 ADDR7 ADDR6 ADDR5 ADDR4 ADDR3      *
ADDR2  ADDR1 ADDR0
END ROM8
$END MODELS
INPUT CLK CE CEINV ALE RDINV IOMINV IORINV            *
ADDR10 ADDR9 ADDR8 ADDR7 ADDR6 ADDR5 ADDR4 ADDR3      *
ADDR2  ADDR1 ADDR0
OUTPUT READY DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
ROM8 *
READY DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 *
CLK CE CEINV ALE RDINV IOMINV IORINV                  *
ADDR10 ADDR9 ADDR8 ADDR7 ADDR6 ADDR5 ADDR4 ADDR3      *
ADDR2  ADDR1 ADDR0
END
```

If SISL is to be used specify:

```
ROM8 READY DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 ; *
     CLK CE CEINV ALE RDINV IOMINV IORINV ADDR10 ADDR9 ADDR8 ADDR7 *
     ADDR6 ADDR5 ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
```

and leave off the SALOGS $MODELS portion.

The ROM model is a FORTRAN subroutine mirroring the functional specifications found in the data book. It takes into account proprietary information which indicates how the device should react given inputs not mentioned in its specification sheets.

THE 256 X 8 RAM

The 256 X 8 RAM was relatively easy to create. A RAM "Random Access Memory" is very much like a ROM except that it can write as well as read. Its information may indeed be pre-loaded but that information is subject to change while the system is running. Unless a new loading process takes place, the RAM will loose its stored data while the ROM will not. [II,45]

Appendix F shows the original version of the RAM, the implemented version, and its operational flowchart.
Appendix F also gives the listing of the RAM model. This model is based on that of the ROM with the added writing feature.

To access the RAM use the following SALOGS code:

```
$MODELS
RAM8  0 8 11 22 9 0
DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 *
RESET WRINV CEINV ALE RDINV IOMINV             *
ADDR7 ADDR6 ADDR5 ADDR4 ADDR3                  *
ADDR2  ADDR1 ADDR0
END RAM8
$END MODELS
INPUT RESET WRINV CEINV ALE RDINV IOMINV                  *
                     ADDR7 ADDR6 ADDR5 ADDR4 ADDR3      *
                     ADDR2  ADDR1 ADDR0
OUTPUT DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
RAM8 *
      DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 *
      RESET WRINV CEINV ALE RDINV IOMINV               *
      .              ADDR7 ADDR6 ADDR5 ADDR4 ADDR3      *
                     ADDR2  ADDR1 ADDR0
END
```

If SISL processing is to be done use:

```
RAM8 DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 ; *
     RESET WRINV CEINV ALE RDINV IOMINV ADDR7 ADDR6 ADDR5 ADDR4 *
     ADDR3 ADDR2 ADDR1 ADDR0
```

and leave off the SALOGS $MODELS portion.

This model also uses the functional specification modeling technique. For further detail on SALOGS and its use of models, refer to the Appendix and the SALOGS USERS GUIDE. See Appendix H for a listing of the commands needed to bring the various SISL and SALOGS software on line. The package is supported to run on the DEC SYSTEM 10 using the TOPS-10/603A96.04 operating system.

CH 7. A REVIEW OF INDIVIDUAL FUNCTIONAL MODELING LANGUAGES

Before describing the SALOGS pre-processor ,SISL, it would be instructive to review some of the languages which have already been created for CAD. A study of languages was undertaken to help decide on the detail required, how a language structure should be defined, and to discover a standard upon which a user interface could be based. The convenience of the user is very important as is the following of commonly accepted standards of circuit modeling. The chosen language must be expandable so that is can remain current with modern technology.

There are many levels at which one may represent a digital system. Vertically, the following levels in order of detail may be chosen:

- electron or physics level
- discrete device
- circuit
- gate
- chip
- functional

SISL models the structure of systems at the functional level and SALOGS simulates systems at the gate level. Together, they let the designer model at the gate, chip, and functional levels all at the same time. Horizontally, one may split the system into many or few modules or subystems. The behavior of the total system may be studied in more or less detail by observing the simulation states which appear on the lines interconnecting the subsystems.

Many languages have been created by others working in the field. These permit the description of exercising of digital hardware at one or more of the horizontal and/or vertical levels.

ISP (Instruction Set Processor) [S3,39]

ISP was developed to describe a computers' programming and register transfer levels. Thus, it can be used to study the behavior of a digital processor. It describes computers by using various fixed formats. Permitted are declarations and actions affecting memory, processor state, primary memory, console state, I/O state, data types, data operations, and instruction formats. Overall, it allows one to model computers at a very high level but does not describe the inner workings of the hardware beyond register transfer operations.

AHPL (A Hardware Programming Language) [H6,28]

AHPL uses the notational conventions of APL (A Programming Language) [11,1] to describe digital hardware. Its utility comes from the partitioning of a system into a control section and a data register/logic section. The control circuit causes register transfers to take place in the data section by putting signals on its control lines. Branching information from the data section influences the sequence of the control signals.

This is a very low level, register transfer language. It does not precisely describe the structure of a digital system but simulates its output based on input. Its simulation is based on the moving of data from place to place and may be said to work at the information level.

PMS (Processors, Memory, and Switches) [S4,42]

PMS will allow the description of computer systems in terms of the physical interconnection of a small number of elementary components. One of its main aims is to create a standard whereby designers may discuss their simulations. It can be used to focus on certain structures or register transfer and switching circuits.

The basic component types are memory, links, controls, switches, transducers, data operations, and processors. These seven components can be connected to create a eighth type called a stored program computer.

Many of the basic component types here are required in the description of digital systems in general. It would be possible to describe the behavior of a specific chip using this language. A problem is that a simulation project would be made much easier if specific elements were available which described the behavior of given devices which can be purchased off the shelf.

FST (Functional Simulator and Translator) [S7,46]

FST gives a designer the ability to specify logical sequences of operations without having to explicitly specify the control logic. Models are described in sequential and concurrent blocks. Any control logic is implicit in the description of a block and is produced by FST itself.

This language presents ideas which are very near to what is desired in the new language. It handles structure and operations separately and can be used at a variety of modeling levels. (A block could be an AND gate or a CPU for instance.)

LALSD (Language for Automated Logic and System Design) [S7,47]

LALSD uses a multi-level modeling approach which allows simulation at any level of detail. Designs are seen to have two parts: structure and control. It is very much like AHPL in that the control section, describing system behavior, sends signals to the structure part to initiate operations.

This language also allows the partitioning of a system into sub-blocks which can be easily integrated. It has facilities for linking subroutines to its base software. Control signals were separated from the structure for the following reasons:

- If a person is only interested in the behavior of a system, it is not necessary to study the structure.
- The control part can be implemented in hardware, firmware, or software. Thus, there is a flexibility which aids economical realization.
- Such a model is very convenient for high-level modeling such as looking at determinancy and deadlocks. Exhaustive simulation is avoided.

The general ideas behind the language are very much in keeping with the goals attempted by this investigation. It will allow working at arbitrary levels and intermixing them in one simulation.

SDL (Structural Description Language) [V2,1]

SDL describes in detail the interconnections of a series of digital blocks. These blocks may be of any modeling level. It will also specify the interfaces between two or more subsystems. Contained in its syntax is the ability to describe node names, block names, interconnections between blocks, numbers of I/O lines, and other specifications used to fully define the construction of a digital system. In short, one could take a schematic and translate its stucture to SDL.

Because of its syntax rules, which allow the easy specification of a system structure, this language could fulfill the requirement for specifying the interconnection of elements which are at first undefined in their behavior. SDL does not intermix behavior modeling with structural modeling.

IN SUMMARY

From the above discussion, the reader will note that several of the above languages meet many of the goals as outlined in the introduction. It remains to combine the best features of each to solve the particular problems at hand. This combination is found in SISL and its interface to SALOGS.

CH 8. CHOOSING A MODELING LANGUAGE

This chapter is devoted to the description of the SALOGS preprocessor, SISL. A view of its inner details will be given along with the basic philosophy behind the language. This will demonstrate its completeness as well as its usefullness to CAD activities.

There are several requirements which have been noted by those who write description languages. These are necessary for the clear and complete specification of a digital system. [D5,1]

1. ability to name and describe blocks which correspond one-to-one with those of the system being designed
2. separation of process and control
3. support for several modeling levels
4. separation of the various phases of simulation and testing
5. allowance of concurrent activities at the several modeling levels
6. specification of synchronous and asynchronous activities
7. description of data routing between elements

To these may be added:

8. support of the user in his attempts to describe a system
9. easy interfacing to other design packages
10. following of standards which are generally accepted in the design community

The choice made for a language to support the intermixing of functional and gate modeling levels was based on the above criteria. SDL was chosen for the basic syntax of structural modeling and SALOGS was chosen for process control. (See Appendix D for a description of SALOGS.) The review of the other languages was used to create SISL which combines syntax features of SDL and SALOGS. It extends the modeling capabilities of SALOGS by making it much easier to use beyond the gate level.

The syntax of register transfer level, programming level, and information level modeling did not appear to be conducive to the intermixing of widely separate design levels. However, the general ideas presented by the other languages were valuable in the effort to derive the new language, SISL.

The details availsable on SDL were sufficient for an in-depth study of that language. Also, its syntax is very close conceptually to such languages as PCAP (Princeton Circuit Analysis Program) [S8,1] which is a discrete component level circuit simulation package. Many practicing engineers began by using similar design aids. An intuitive feel for SDL's use can be easily developed since it is "natural" to a human user. SDL's syntax works very hard for the designer.

Thus, a subset of SDL was chosen to begin the construction of SISL's syntax. It was modified slightly to conform more closely to that of SALOGS and to cover some areas that might help the user make fewer errors when describing a system. (More on this later.)

SISL DETAILS

SISL itself has no ability to define a process, that resides in SALOGS. (*) It lets one describe the structure of arbitrary (functional) level digital systems and their interface to the gate level portion of those systems. SISL simply adds to SALOGS the ability to easily describe structures of a functional level in addition to its gate level without having to do FORTRAN coding or to become involved with the details of SALOGS' $MODELS section. (See the SALOGS USERS GUIDE.)

The SALOGS/SISL system separates process and control. The process is the behavioral model of the functional element written in FORTRAN. Control resides in the structure of the digital system. A behavioral model may be changed at will (as long as the number of I/O lines remains the same) without the need to modify the system structure.

(*) For further detail refer to the SISL USERS GUIDE.

Element names may be anything the designer chooses as long as the basic naming syntax is followed. This package is modular in that it has the following [illegible] level [illegible], [illegible] level structure, routines to compile the combination of functional and gate level structures, routines to compile the exercising commands, routines to perform the exercising commands, and routines to perform fault analysis.

SISL SYNTAX

Appendix B shows the syntax of SISL. There are two differences between it and that of SALOGS.

- A ";" separates the list of output nodes and input nodes. This is to force the user to carefully consider line assignments. When one may specify up to 40 nodes per element, this becomes necessary. Also, it provides an aid to the user proofing of the structural description.

- A "*" as the first character rather than only in column #1 flags a comment line. This is a user convenience and allows creation of banners without the need for an extraneously filled column.

The syntax rules are not as extensive as that for SDL, the model for SISL, since only the interconnection of pre-defined elements is considered. However, SISL is complete and will allow the description of system structures where the elements contain up to 40 I/O lines each. This limit had to be set due to SALOGS' internal restrictions. SISL is designed to interface easily with SALOGS.

USER INTERFACE

SISL assists the user through its carefully designed syntax rules, though its ease of interface to SAGOGS, and through its user proofing. It is friendly and does not take long to learn. Also, being modular in its construction, modifications and additions are not difficult to make.

User proofing is perhaps the most important feature of a package which is meant for release to those who have no need to understand the inner workings of the software. Basically, a philosophy of error checking should include the detection of problems at the earliest possible point in the program. Errors should not be allowed to propagate beyond their point of earliest detectability. Too,

> The user must be able to determain whether failure of an attempted operation was due to improper control signals or system malfunction [P3,13].

If "...improper control signals..." is replaced by "...improper user input data...", an idea is obtained as to how to approach the delivery of error messages.

While SISL will not catch all conceivable user errors, it will note errors due to syntax violations and inconsistencies. These include a node being used for input but not for output and an incorrect number of I/O nodes for an element.

TRANSLATION [illegible]

SISL [illegible] and converts the syntax to that required by the SALOGS $MODELS portion. This can be a rather extensive conversion since SALOGS requires quite a lot to set up a structure at the functional level. (See Appendix A for an example.)

No node name conversions are made although SISL will check the correspondance of numbers of I/O lines and the naming conventions. It will also ensure that each node is used at least once for input and for output. During the parsing of the several syntax diagrams, SISL will check the integrity of each. Any error will result in a message and an immediate controlled termination. The syntax diagramming, which guides the parsing of user input, is based on that for the computer language, PASCAL [J2,1]. The procedure is the author's original design Each line of input data is dealt with as a single entity. It is not necessary for SISL to know what went before or what comes later. A lines' syntax is translated and then spooled to a scratch file which eventually becomes the $MODELS heading for the SALOGS gate level portion of the overall system description.

Node names are retained as is so that throughout a complete simulation, the designer will not be troubled with several words which stand for the same thing. The intent has been to ease the burden placed on the user during the design process.

CH 9. PROJECT SUMMARY AND CONCLUSIONS

The objective of [illegible] was [illegible] gate and functional level digital simulations. This goal was met by the creation of SISL, a functional level preprocessor to the gate level SALOGS CAD package, and a library of three behavioral models. The state of the art has been advanced because the sponsors will now be able to support projects previously denied due to the D*C*K modeling limitations. An easy mix of functional and gate level modeling has been achieved. A designer may directly intermix the two levels of modeling while saving main memory and time. The assumption here is that a behavioral model of a system element will perform faster and in less main memory than its gate level counterpart. It will deliver less detail, but the extra detail is not desired by most designers modeling at levels beyond the gate level.

The group of possible users include all of the institutions presently making use of SALOGS. These include several universities and industrial concerns as well as the two sponsors.

SISL is presently running on the AVIONICS LABS DEC SYSTEM 10 and the SANDIA LABS DEC SYSTEM 20. It has proven itself a great aid in the modeling of digital systems. Continuing support will be carried out by the author (see VITA for address) through SANDIA LABS.

Any individual wishing to extend this study should look to the creation of additional behavioral [illegible] will interface to [illegible]. The value of any modeling [illegible] on the [illegible] and types of [illegible] to it. The more [illegible], the more a user will be willing to learn and use a design aid. An extensive library would mean that FORTRAN coding would not have to be done to include an element in a system model. The timing question should also be addressed. As digital systems become faster and faster, the timing issue increases in importance.

Presently, only the following are in the behavioral library: a four input OR gate, 4-16 decoder, 2K X 8 ROM, and a 256 X 8 RAM. An ALU would be encouraged by the sponsors as would a CPU. An extensive timing buss would also be valuable.

Anyone wishing to use or extend the features of SISL or its behavioral library should feel free to contact the author or the sponsors.

BIBLIOGRAPHY

A1 ACKEN, JOHN M. AND JERRY D. STAUFFER
"[illegible] OF CIRCUIT SIMULATION," IEEE CIRCUITS AND SYSTEMS MAGAZINE,
1: 9-1[illegible] (JAN 79)

A2 ACKEN, JOHN M. AND JERRY D. STAUFFER
"PART 2:LOGIC CIRCUIT SIMULATION," IEEE CIRCUITS AND SYSTEMS MAGAZINE,
1: 3-12 (FEB 79)

A3 ADKINS, GERALD AND UDO W. POOCH
"COMPUTER SIMULATION: A TUTORIAL," COMPUTER, 12-17 (APR 77)

B1 BOLLIG, C.J.; C.R. GORDON; M.A. WIELAND. "FUNCTIONAL TESTING OF
RANDOM LOGIC USING SIMULATION DATA," IBM TECHNICAL DISCLOSURE
BULLETIN, 20 #5: 1916-1917 (1977)

B2 BOWERS, JAMES C. AND STEPHEN R. SEDORE.
SCEPTRE...A COMPUTER PROGRAM FOR CIRCUIT AND SYSTEMS ANALYSIS.
ENGLEWOOD CLIFFS, NJ: PRENTICE-HALL INC 1971

B3 BOYD, D.L. AND A. PIZZARELLO "INTRODUCTION TO THE WELLMADE
DESIGN METHODOLOGY," IEEE TRANSACTIONS ON SOFTWARE
ENGINEERING, SE-4: 276-282 (JUL 78)

C1 CARR, WILLIAM N. AND JACK P. MIZE. *MOS/LSI DESIGN AND APPLICATIONS*.
NEW YORK, NY: MCGRAW-HILL BOOK CO 1972

C2 CASE, GLENN R.
SALOGS-A CDC6600 PROGRAM TO SIMULATE DIGITAL LOGIC NETWORKS.
ALBUQUERQUE,NEW MEXICO: SANDIA LABORATORIES, 1977.

C3 CHANG, H.Y. AND S.G. CHAPPELL
"DEDUCTIVE TECHNIQUES FOR SIMULATING LOGIC CIRCUITS,"
COMPUTER, 52-59 (MAR 75)

C4 CHATTERGY, RAHUL AND UDO W. POOCH
"INTEGRATED DESIGN AND VERIFICATION OF SIMULATION PROGRAMS,"
COMPUTER, 40-45 (APR 77)

C5 CHU, YAOHAN "WHY DO WE NEED COMPUTER HARDWARE DESCRIPTION
LANGUAGES," COMPUTER, 18-22 (DEC 74)

C6 CHU, YAOHAN "INTRODUCING CDL," COMPUTER, 31-33 (DEC 74)

C7 CIAMPI, P.L.; A.D. DONOVAN; J.D. NASH "CONTROL AND
INTEGRATION OF A CAD DATA BASE," PROCEEDINGS: DESIGN
AUTOMATION CONFERENCE, 13: 285-289 (JUN 76)

C8 CIAMPI, P.L.; J.D. NASH "CONCEPTS IN CAD DATA BASE
STRUCTURES," PROCEEDINGS: DESIGN AUTOMATION CONFERENCE,
13: 290-294 (JUN 76)

C9 CORTNER, MAX J. "A USERS' VIEW OF THE NEXT STEP IN TEST
GENERATION SOFTWARE," PROCEEDINGS: AUTOTESTCON, 300-304 (1979)

D1 DERVISOGLU, BULENT "HARDWARE DESCRIPTION LANGUAGES IN GREAT BRITAIN," COMPUTER, 64-66 (DEC 74)

D2 DIETMEYER, D.L. "INTRODUCING DDL," COMPUTER, 34-38 (DEC 74)

D3 DIGITAL [illegible]
MAYNARD, MASS: DIGITAL EQUIPMENT CORP 1974

D4 DIGITAL EQUIPMENT CORP. GETTING STARTED WITH DECSYSTEM 10.
MAYNARD, MASS: DIGITAL EQUIPMENT CORP 1975

D5 DUDANI, S. AND E.P. STABLER. "REPORT ON A SURVEY OF HARDWARE DESCRIPTION LANGUAGES,"
UNPUBLISHED, ROME AIR DEVELOPMENT CENTER (1978)

D6 DZIDA, W.; S. HERDA; W.D. ITZFELDT "USER PERCEIVED QUALITY OF INTERACTIVE SYSTEMS," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, SE-4 #4: 270-276 (JUL 78)

F1 FREEMAN, PETER "SOFTWARE RELIABILITY AND DESIGN: A SURVEY," PROCEEDINGS: DESIGN AUTOMATION CONFERENCE, 13: 484-495 (JUN 76)

F2 FOSTER, J.C. "THE EVOLUTION OF AN INTEGRATED DATABASE," PROCEEDINGS: DESIGN AUTOMATION CONFERENCE, 12: 394-398 (JUN 75)

G1 GARROCQ, CARLOS A. AND MICHAEL J. HURLEY "THE IPAD SYSTEM: A FUTURE MANAGEMENT/ENGINEERING/DESIGN ENVIRONMENT," COMPUTER, 23-33 (APR 75)

G2 GLESNER, M. "A NEW EFFICIENT CIRCUIT APPROACH FOR MACROMODELING DIGITAL CIRCUITS," PROCEEDINGS: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 359-363 (1978)

G3 GLESNER, MANFRED. "NEW MACROMODELING APPROACHES FOR THE SIMULATION OF LARGE SCALE INTEGRATED CIRCUITS," PROCEEDINGS: EUROPEAN CONFERENCE ON CIRCUIT THEORY AND DESIGN, 448-452 (1978)

G4 GREEN, DAVID. "LARGE SCALE INTEGRATION SIMULATION FACILITY," UNPUBLISHED, SANDIA LABS (8 APR 80)

G5 GROEGER, H.J. "A NEW APPROACH TO STRUCTURAL PARTITIONING OF COMPUTER LOGIC," PROCEEDINGS:DESIGN AUTOMATION CONFERENCE, 12: 378-383 (JUN 75)

H1 HAYTER, R.B.; P.S. WILCOX; H. ROMBEEK; D.M. CAUGHEY. "STANDARD CELL MACROMODELS FOR LOGIC SIMULATION OF CUSTOM LSI," PROCEEDINGS: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 1108-1112 (1978)

H2 HEMMING C.W. AND J.M. HEMPHILL "DIGITAL LOGIC SIMULATION MODELS AND EVOLVING TECHNOLOGY," PROCEEDINGS: DESIGN AUTOMATION CONFERENCE, 12: 85-94 (JUN 75)

H3 HENCKELS, LUTZ; RENE HAAS; CHI-YUAN LO "HIERARCHICAL AND CONCURRENT FAULT SIMULATION," ELECTRONICS TEST, 54-58 (MAR 80)

H4 HEYDEMANN, M.H. "FUNCTIONAL MACROMODELING OF ELECTRICAL CIRCUITS," PROCEEDINGS: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 532-535 (1978)

H5 HILL, FREDERICK J. AND GERALD R. PETERSON.
DIGITAL SYSTEMS: HARDWARE ORGANIZATION AND DESIGN.
NEW YORK, NY: JOHN WILEY AND SONS INC 1973

H6 HILL, FREDERICK J. "INTRODUCING AHPL," COMPUTER, 28-30 (DEC 74)

H7 HOWDEN, WILLIAM E. "THEORETICAL AND EMPERICAL STUDIES OF PROGRAM TESTING," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, SE-4 #4: 293-298 (JUL 78)

H8 HSUEH, M.Y.; A.R. NEWTON; D.O. PEDERSON. "THE DEVELOPMENT OF MACROMODELS FOR MOS TIMING SIMULATORS," PROCEEDINGS: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 345-349 (1978)

I1 IBM CORP. SYSTEMS: APL LANGUAGE. IBM: SAN JOSE, CA 1978

I2 INTEL CORP. MCS-48 MICROCOMPUTER USERS MANUAL.
SANTA CLARA, CALIF: INTEL CORP 1978

J1 JACQUART R.; PH. REGNIER; F.R. VALETTE; J. POISSEAU.
"CURRENT TRENDS IN THE DEVELOPMENT OF INTEGRATED GENERAL PURPOSE CAD SYSTEMS," PROCEEDINGS: DESIGN AUTOMATION CONFERENCE, 12: 180-188 (JUN 75)

J2 JENSEN, KATHLEEN AND NIKLAUS WIRTH. PASCAL-USER MANUAL AND REPORT.
NEW YORK, NY: SPRINGER-VERLAG 1978

L1 LANCASTER, DON. CMOS COOKBOOK. INDIANAPOLIS, IND:
HOWARD W. SAMS AND CO 1977

M1 MARCOZ, F. AND O. TEDONE "HARDWARE DESCRIPTION LANGUAGES IN ITALY," COMPUTER, 60-61 (DEC 74)

M2 MERMET, JEAN "HARDWARE DESCRIPTION LANGUAGES IN FRANCE," COMPUTER, 55-56 (DEC 74)

M3 MERWIN, RICHARD E. "SOFTWARE MANAGEMENT: WE MUST FIND A WAY," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, SE-4 #4: 307-308 (JUL 78)

N1 NATIONAL SEMICONDUCTOR. CMOS DATA BOOK.
SANTA CLARA, CALIF: NATIONAL SEMICONDUCTOR 1977

P1 PEDERSEN, JAN T.; JOHN K. BUCKLE "KONGSBERG'S ROAD TO AN INDUSTRIAL SOFTWARE METHODOLOGY," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, SE-4 #4: 263-269 (JUL 78)

P2 PILOTY, ROBERT "HARDWARE DESCRIPTION LANGUAGES IN THE FEDERAL REPUBLIC OF GERMANY," COMPUTER, 57-59 (DEC 74)

P3 PURVIS, RICHARD E. AND RONALD D. YOHO. MIME: MICROPROGRAMMABLE MINICOMPUTER EMULATOR. UNPUBLISHED THESIS:
AIR FORCE INSTITUTE OF TECHNOLOGY 1978

R1 RAETH, PETER G. AND WILLIAM J. BARKSDALE
"AN INTERACTIVE COMPUTER PACKAGE FOR ACTIVE FILTER DESIGN,"
PROCEEDINGS: IEEE SOUTHEASTCON, 351-354 (APR 78)

R2 RAETH, PETER; [illegible]; [illegible]; JOHN [illegible]
"[illegible],"
UNPUBLISHED, SANDIA LABS (NOV 80)

R3 RALSTON, ANTHONY AND C.L. MEEK ENCYCLOPEDIA OF COMPUTER SCIENCE.
PETROCELLI/CHARTER: NEW YORK, NY 1976

S1 SCHNEIDERWIND, N.F. "THE USE OF SIMULATION IN THE EVALUATION OF
SOFTWARE," COMPUTER, 47-43 (APR 77)

S2 SHERWOOD, WILL. "THE BOOLEAN CONNECTION," PROCEEDINGS:
SEMICONDUCTOR TEST CONFERENCE, 274-281 (1978)

S3 SIEWIOREK, DAN "INTRODUCING ISP," COMPUTER, 39-41 (DEC 74)

S4 SIEWIOREK, DAN "INTRODUCING PMS," COMPUTER, 42-44 (DEC 74)

S5 SOONG, NORMAN LOONGSUNG "THE DESIGN OF PROGRAM LOGIC,"
PROCEEDINGS: DESIGN AUTOMATION CONFERENCE,
13: 354-365 (JUN 76)

S6 STUTZ, E.A. AND W.M. VAN CLEEMPUT. "STRUCTURAL INTEGRATED CIRCUIT
DESIGN WITH THE HIDRIS SYSTEM," PROCEEDINGS: NATIONAL
ELECTRONICS CONFERENCE, 123-128 (1978)

S7 SU, STEPHEN Y.H. "A SURVEY OF COMPUTER HARDWARE DESCRIPTION
LANGUAGES IN THE USA," COMPUTER, 45-51 (DEC 74)

S8 SURBER, WILLIAM H. PCAP: PRINCETON CIRCUIT ANALYSIS PROGRAM.
PRINCETON UNIVERSITY: PRINCETON, NJ 1976

S9 SZYGENDA, S.A. "DIGITAL SYSTEMS SIMULATION," COMPUTER, 23-23 (MAR 75)

S10 SZYGENDA, S.A. AND E.W. THOMPSON "DIGITAL LOGIC SIMULATION IN
A TIME-BASED, TABLE-DRIVEN ENVIRONMENT: PART 1. DESIGN
VERIFICATION," COMPUTER, 24-36, (MAR 75)

T1 TANENBAUM, ANDREW S. STRUCTURED COMPUTER ORGANIZATION.
ENGLEWOOD CLIFFS, NJ: PRENTICE-HALL INC 1976

T2 THOMPSON, E.W. AND N. BILLAWALA "THE SOFTWARE ENGINEERING TECHNIQUE
OF DATA HIDING AS APPLIED TO MULTI-LEVEL MODEL IMPLEMENTAION
OF LOGICAL DEVICES IN DIGITAL SIMULATION,"
PROCEEDINGS: DESIGN AUTOMATION CONFERENCE, 12: 195-201 (JUN 75)

T3 THOMPSON, E.W. AND S.A. SZYGENDA "DIGITAL LOGIC SIMULATION IN
A TIME-BASED, TABLE-DRIVEN ENVIROMMENT: PART 2. PARALLEL FAULT
SIMULATION," COMPUTER, 38-49 (MAR 75)

T4 TRACY, JAMES H. "DIFFICULTIES IN APPLYING REGISTER TRANSFER
LANGUAGES IN MICROCOMPUTER SYSTEM DESIGN," PROCEEDINGS:
MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS, 158-161 (1978)

T5 TROXEL,, DONALD E. GRAPHICAL INPUT METHODOLOGY FOR COMPUTER AIDED ANALYSIS OF CONTROL SYSTEMS. UNPUBLISHED THESIS: AIR FORCE INSTITUTE OF TECHNOLOGY 1978

V1 VAN CLEEMPUT, W.M. "DESIGN AUTOMATION TOOLS FOR STRUCTURED HARDWARE DESIGN," [illegible], 1-11 (1979)

V2 VAN CLEEMPUT, W.M. "SDL, A LANGUAGE FOR DESCRIBING THE STRUCTURE OF DIGITAL SYSTEMS," UNPUBLISHED, STANFORD UNIVERSITY (MAR 1979)

V3 VAN CLEEMPUT, W.M. TUTORIAL: COMPUTER-AIDED DESIGN TOOLS FOR DIGITAL SYSTEMS. NEW YORK, NY: IEEE 1970

V4 VAUCHER, JEAN "HARDWARE DESCRIPTION LANGUAGES IN CANADA," COMPUTER, 53-54 (DEC 74)

W1 WATANABE, HITOSHI AND KIICHI FUJINO "HARDWARE DESCRIPTION LANGUAGES IN JAPAN," COMPUTER, 62-63 (DEC 74)

Appendix A

SISL USERS GUIDE

TABLE OF CONTENTS

USERS GUIDE FOR SISL

SISL, Structural Interface to the SALOGS Language, is designed as a functional level preprocessor to SALOGS (SAndia LOGic Simulator) which is a gate level digital simulator. SALOGS simulates digital systems using AND, OR, and INVERT primitives. These primitives or gates perform during the simulation almost the same as do their MOS technology counterparts. Eight signal states are used to partition voltage levels. These eight states include the logical states. Refer to Appendix D of of this guide for more on the application of and the terms applied in in relation to SALOGS.

The purpose of SISL is to allow the description of a digital system in terms of high level devices such as adders, shift registers, etc. rather than in gates such as AND, OR, etc. This level of description will be refered to in this manual as macro descriptions. The term gate level simulation is used here to refer to digital simulation using MOS technology behavior models of AND, OR and INVERT eight state primitives. The several algorithms attendant to SISL perform their preprocessing by implementing a digital system description language very close to that of SALOGS itself. Thus, the designer will find the use of this new level of modeling quite easy to transition to if he has gained a familiarity with the SALOGS design aid.

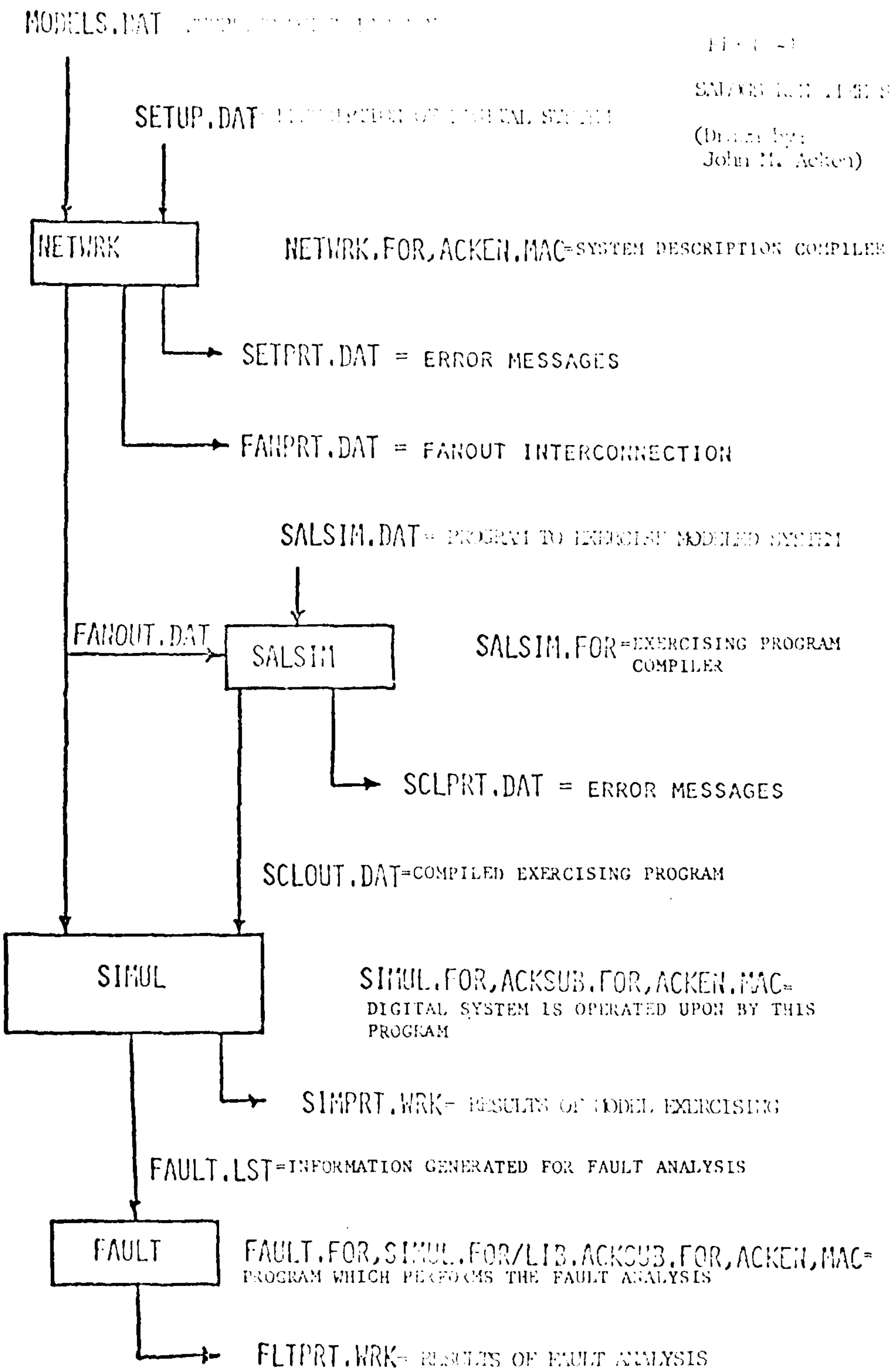
Not only can the designer work at a very high level of system description but he can link a gate level SALOGS model to a SISL macro model and run the entire network as one system. This quide assumes the users' knowledge of SALOGS. Appendix D of this quide contains a copy of the most recent users guide to that package.

OVERVIEW

SISL is designed to run as a functional level preprocessor to the gate level SALOGS modeling software. It will accept functional level descriptor information and return the SALOGS parameters required to create a single entity from the functional and gate level portions of a digital systems' model.

Fig UG-1 shows the run time data and control flow for SALOGS. Obviously, this design software runs as a series of batch programs interfaced through a number of disk files. Fig UG-2 shows how SISL is an added program (preprocessor) to the SALOGS series. The user creates two model files. One holds the SISL macro model portion of a digital system and the other holds the gate level portion. These two files are manipulated by SISL to produce a total network description to be processed by SALOGS. SISL does not produce gate level models for the large scale devices. Rather, it creates linkages to FORTRAN IV behavior models. These behavior models (called functional models in the SALOGS literature) determine how the device operates and the technology involved. They are not required by SISL itself.

(Drawn by:
John M. Acken)



Single lines are data flow.

Double lines are control flow.

SETUP.DAT= SALOGS gate level description. SISL adds on the SALOGS functional and logical models to this file. This file is sent on to SALOGS.

SISL.DAT= The description of the functional level digital system.

SISL.NAM= A list of device names and their individual number of I/O lines and internal subroutine names.

SISL.OUT= All messages generated by SISL during its last run.

```
                              USER INITIATION

                                    | |
                                    | |
                                    | |
                                    | |
                                    | |
                         ---------------------------
SETUP.DAT----------->    |                         |
                         |                         |  ------> SISL.OUT
SISL.NAM------------>    |          SISL           |
                         |                         |  ------> SETUP.DAT
SISL.DAT------------>    |                         |
                         ---------------------------
                                    | |
                                    | |
                                    | |
                                    | |
                                    | |
                         ---------------------------
                         |                         |
                         |                         |
SETUP.DAT --------->     |         SALOGS          |
                         |                         |
                         |                         |
                         ---------------------------
```

FIG UG-2

SISL RUN TIME SYSTEM

SISL SYNTAX

SISL is [illegible] system description language similar to that developed by W.M. Van Cleemput for his SDL or Structural Description Language [V2,1]. For all the seeming complexity, the source code for SISL is only 1100 lines of FORTRAN IV. Appendix C of this guide contains the listing of the software. Appendix B of this guide gives the set of syntax diagrams which completely define this language. One need only study these constructs to understand the syntax.

AN EXAMPLE

To demonstrate the use of SISL, a complete run will now be presented. Fig UG-3 is an example of the block diagram of a digital system. In the next section examples will be given of each file required to program it.

There are several steps required to build a functional/gate level model. These steps are:

1. obtain the gate diagram for the gate model
2. code this model in the SALOGS gate level language
3. test for compile and execution errors of this model
4. decide on the functional level additions to the gate level model
5. write the SISL functional level portion of the overall model
6. test this portion for compile errors
7. run a test on the total functional/gate model
8. repeat steps 1-7 until results are satisfactory

The gate level portion of this system consists only of an AND gate. Since the designer is assumed to already have some expertise with SALOGS, we will only discuss the SISL requirements of the system.

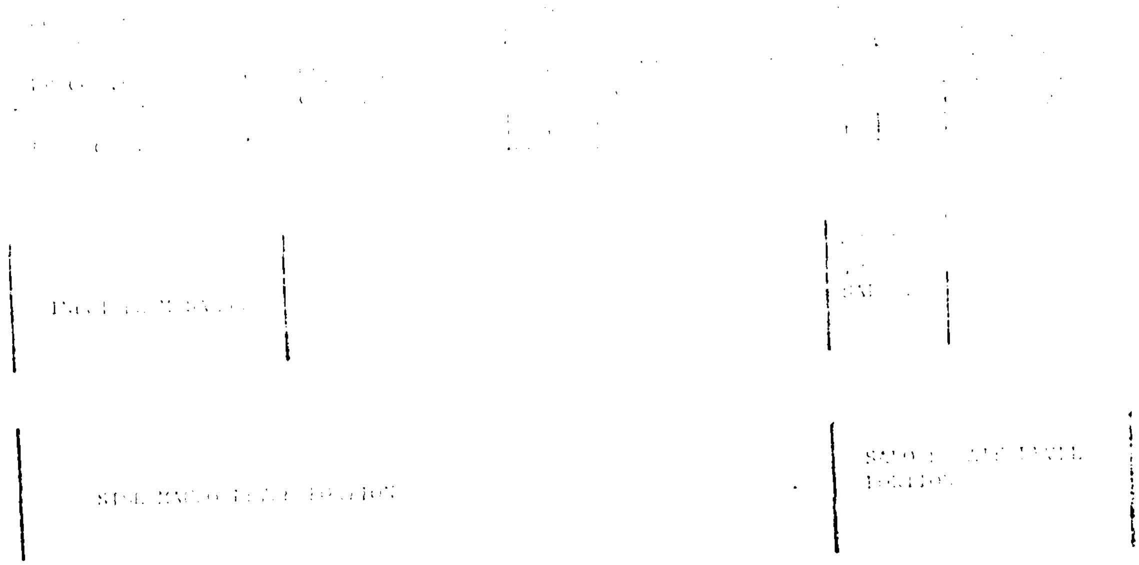


FIG. [illegible]C-3

A[illegible]

SISL INPUT FILES

Three files are required as input to the SISL preprocessor. These files provide the information required by SISL at run time. Following is a list of those files, descriptions of their required format, and a sample of each. Together, these file demonstrate the programming of the block diagram given in Fig UG-3.

SETUP.DAT — The SALOGS gate level portion of the intended digital system; SISL adds to this file the linkages to any required behavior models. Refer to *SALOGS USERS GUIDE* for the details of gate level modeling. SISL assumes that this file originally contains no $MODELS section.

Original version (created by user):

```
INPUT A B C
OUTPUT K
AND K H I J
CIRCUIT H I J A B C
END
```

Final version (created by SISL):

```
$MODELS
COUNTUD     0     3     3     6     2     0
F E D   A B C
END COUNTUD
BTOG        0     3     3     6     1     0
H I J   F E D
END BTOG
CIRCUIT    2     3     3     6     0     0
H       I        J        A        B        C
COUNTUD F E D   A B C
BTOG H I J   F E D
END CIRCUIT
$END MODELS
INPUT A B C
OUTPUT K
AND K H I J
CIRCUIT H I J A B C
END
```

SISL.DAT The description of the macro portion of the digital [illegible] gates. Each device used in the network is mentioned along with its attendant I/O lines in this fashion:

DEVICENAME(SPACE)OUTLIST(SPACE);(SPACE)INLIST where OUTLIST is the list of nodes forming the output from the device and INLIST is the list of inputs to the device.

The first non-comment line of this file is: CONNECT(SPACE)OUTLIST(SPACE);(SPACE)INLIST where OUTLIST is the list of macro model outputs to the gate model and INLIST is the list of inputs coming from the gate model.

```
*
 *
  *
   * THE SECOND TEST OF THE SISL/SALOGS TRANSLATOR
  *
 *
*
CONNECT H I J ; A B C
COUNTUD F E D ; A B C
BIGG H I J ; F E D
END
```

Continuations are noted by using a space followed by a "*" at the end of the continued line. A line may be continued from any point where a space occurs. Comment lines are noted by having a "*" as the first character of a line. An example of a continuation follows:

```
CONNECT H I J ; *
        A B C
```

SISL.NAM A list of all the allowed high level devices and certain parameters unique to each one. Every high level device usable by SISL and having code in the behavioral library is listed in this way:

LINE 1-- Devicename(col 1-8), left justified to column one

LINE 2-- number output nodes required (col 1-5)
number input nodes required (col 6-10)
SALOGS functional model number (col 11-15).
All numbers are integer and right justified.

```
BTOG
    3    3    1
COUNTUD
    3    3    2
```

SISL OUTPUT FILES

SETUP.DAT The combination of SALOGS gate, functional, and logical models created by SISL. This file contains the total system to be simulated and is passed on to SALOGS.

SISL.OUT All messages generated by SISL during its last run. Each message is of the format:

SUBROUTINE GENERATING MESSAGE, FORMAT NUMBER, and MESSAGE

Fig UG-4 gives an example of this file.

```
READFM 5--
BLOCK NAME        # OUTPUTS       # INPUTS       # LINES       ENUM?       HASH #
BTOG                      3              3             6           1            5
[illegible]               3              3             6           2           11

LINEIN 15-- *
LINEIN 15--  *
LINEIN 15--   *
LINEIN 15--    * THE SECOND TEST OF THE SISL/SALOGS TRANSLATOR
LINEIN 15--   *
LINEIN 15--  *
LINEIN 15-- *
LINEIN 15-- CONNECT H I J ; A B C

CONECT 410--
TOTAL # NODES COMMON TO SALOGS=          6
OUTPUT NODES TO SALOGS=                  3
INPUT NODES FROM SALOGS=                 3


*** OUTLIST ***
H
I
J



*** INLIST ***
A
B
C

GETMOD 2-- AM BUILDING THE SALOGS FUNCTIONAL MODELS
LINEIN 15-- COUNTUD F E D ; A B C
LINEIN 15-- BTOG H I J ; F E D
LINEIN 15-- END

GETMOD 1002--
NODE NAME       OUTFLAG       INFLAG       HASH #
A                     1            1           20
B                     1            1           21
C                     1            1           22
D                     1            1           23
E                     1            1           24
F                     1            1           25
H                     1            1           27
I                     1            1           28
J                     1            1           29

LMODEL 5-- AM BUILDING THE SALOGS LOGICAL MODEL
ENDMOD 3-- AM REBUILDING SETUP.DAT FOR SALOGS
```

The 1 under OUTFLAG/INFLAG indicates that the node has been used as an output/input node.

FIG UG-4 AN EXAMPLE OF SISL.OUT

SISL TEMPORARY FILES

SISL uses two files, TEMP1 and TEMP2, for working storage. these files are created and deleted during any given run.

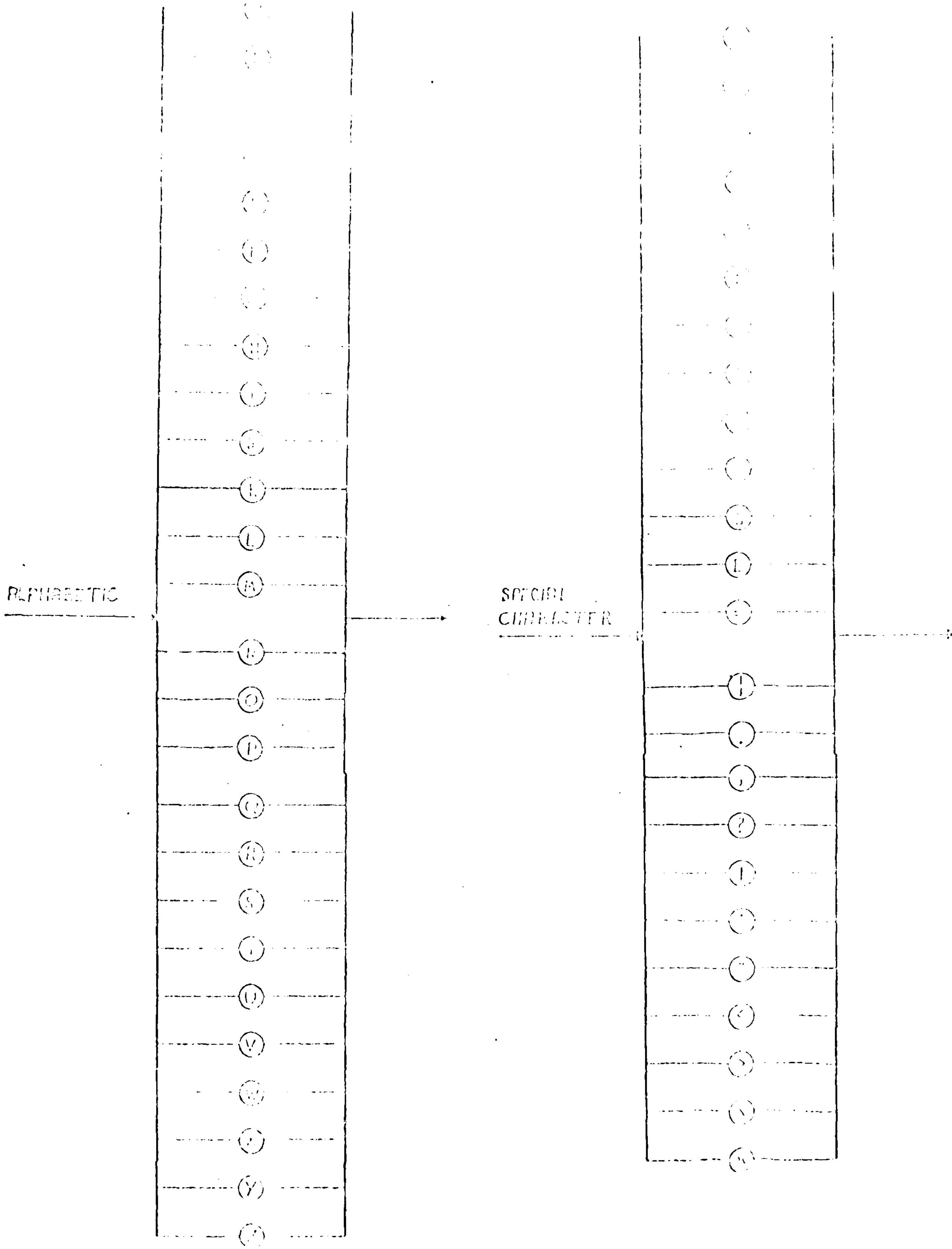
CAUTION ON FILES

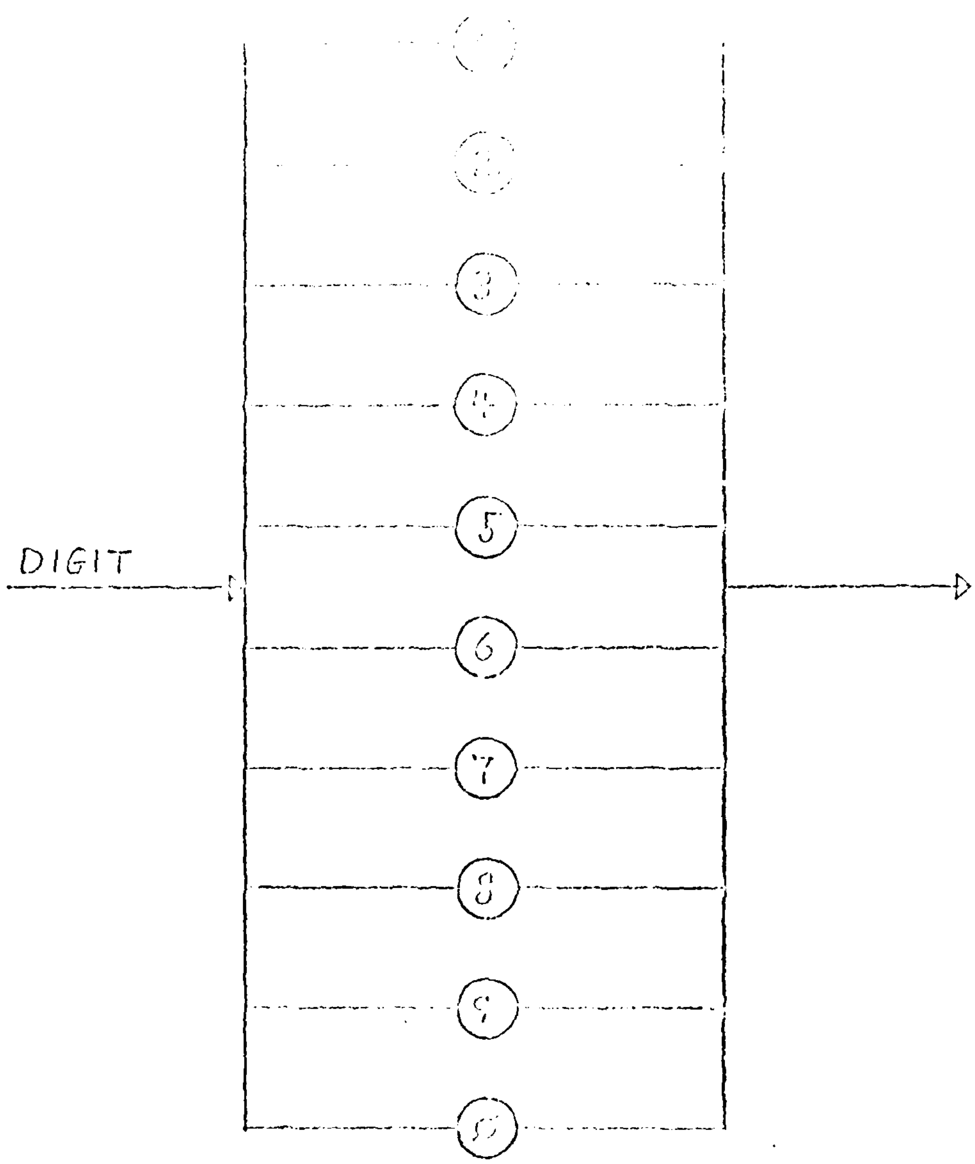
Any file used for output by SISL should always be backed up by the user prior to each run if the current version of that file is desired for retention. This is particularly true of SETUP.DAT since it is used first by SISL as the gate level portion of the digital network and then to contain the total system description.

APPENDIX B

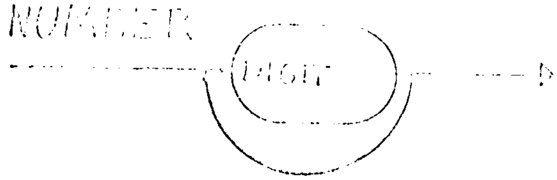
SISL SYNTAX DIAGRAMS

TABLE OF CONTENTS

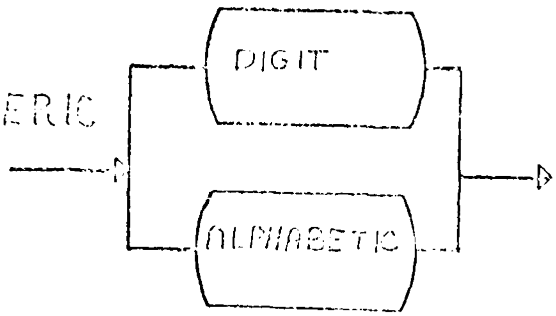

ALPHABETIC
SPECIAL CHARACTER


DIGIT
3
4
5
6
7
8
9
0

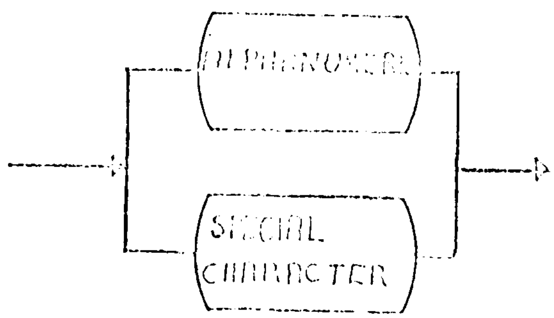
NUMBER



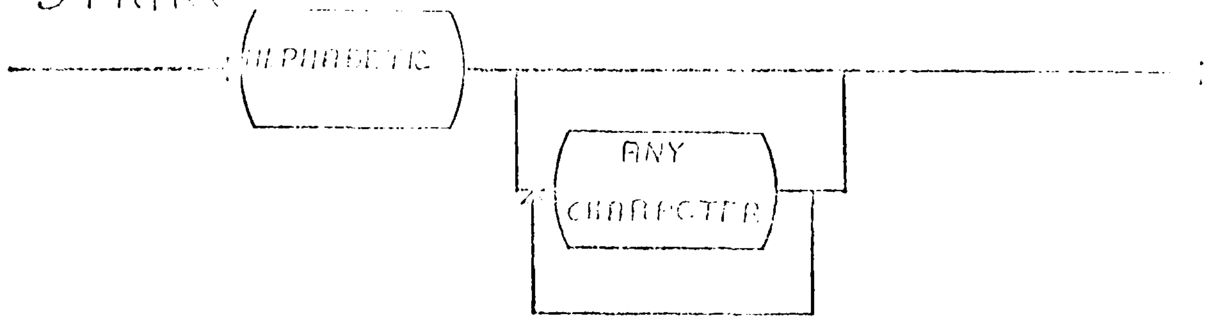
ALPHANUMERIC



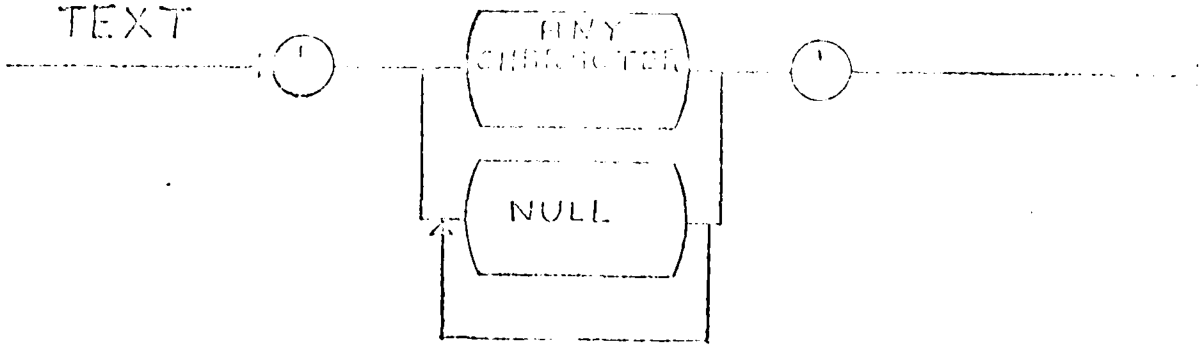
ANY CHARACTER

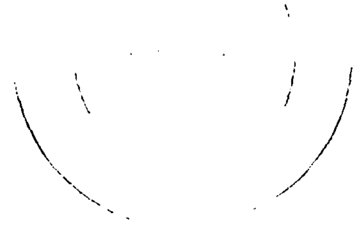


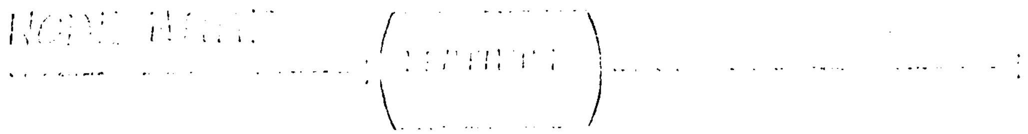
STRING



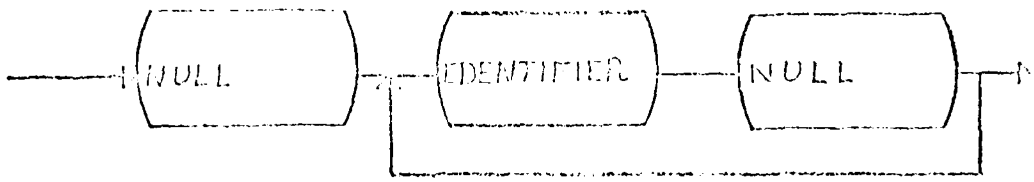
TEXT



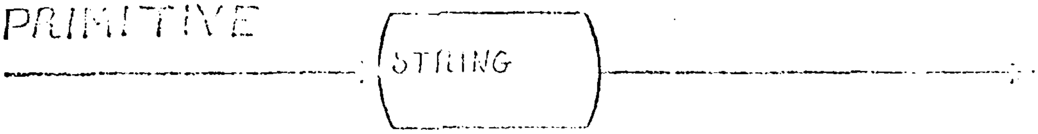




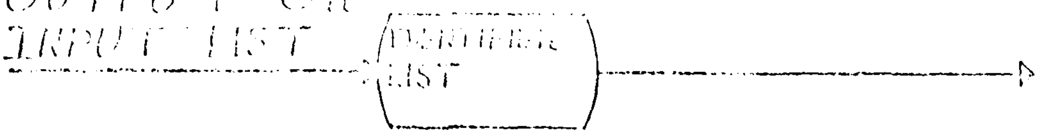
IDENTIFIER LIST

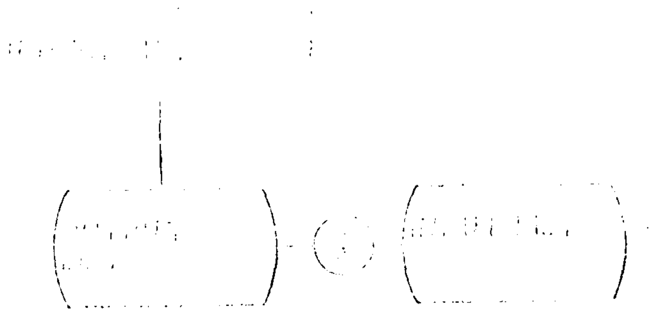


NAME OF FUNCTIONAL
PRIMITIVE

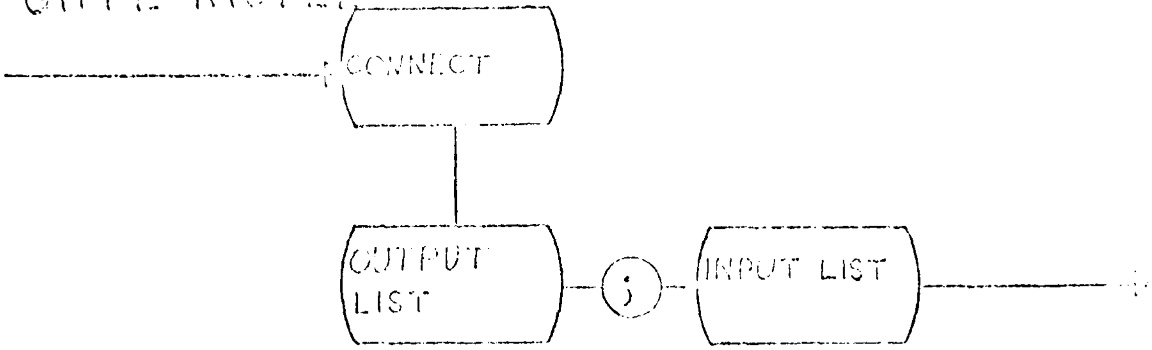


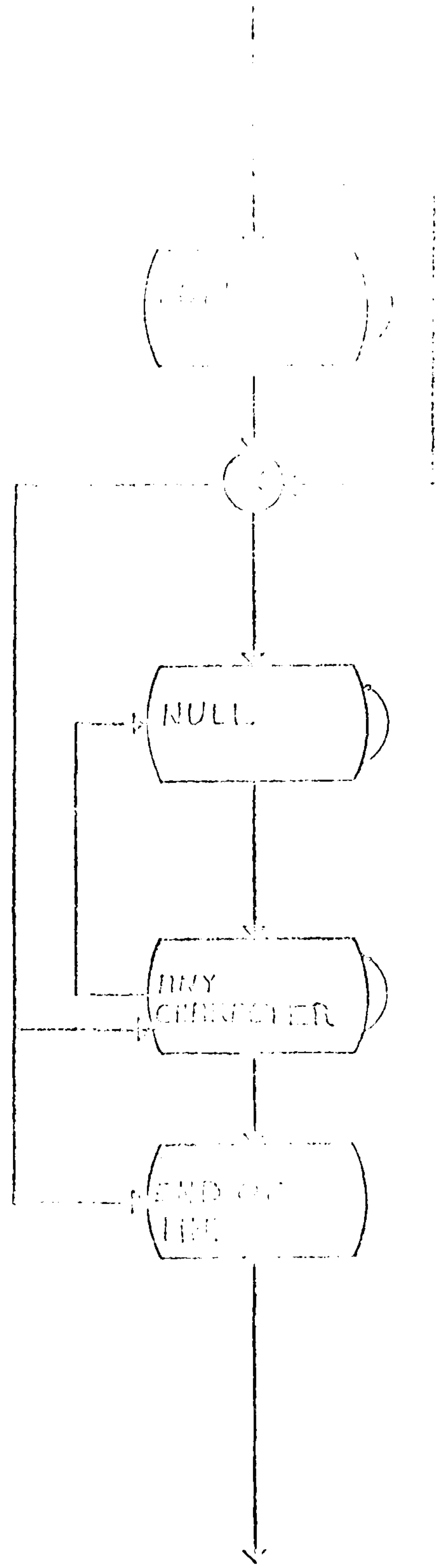
OUTPUT OR
INPUT LIST

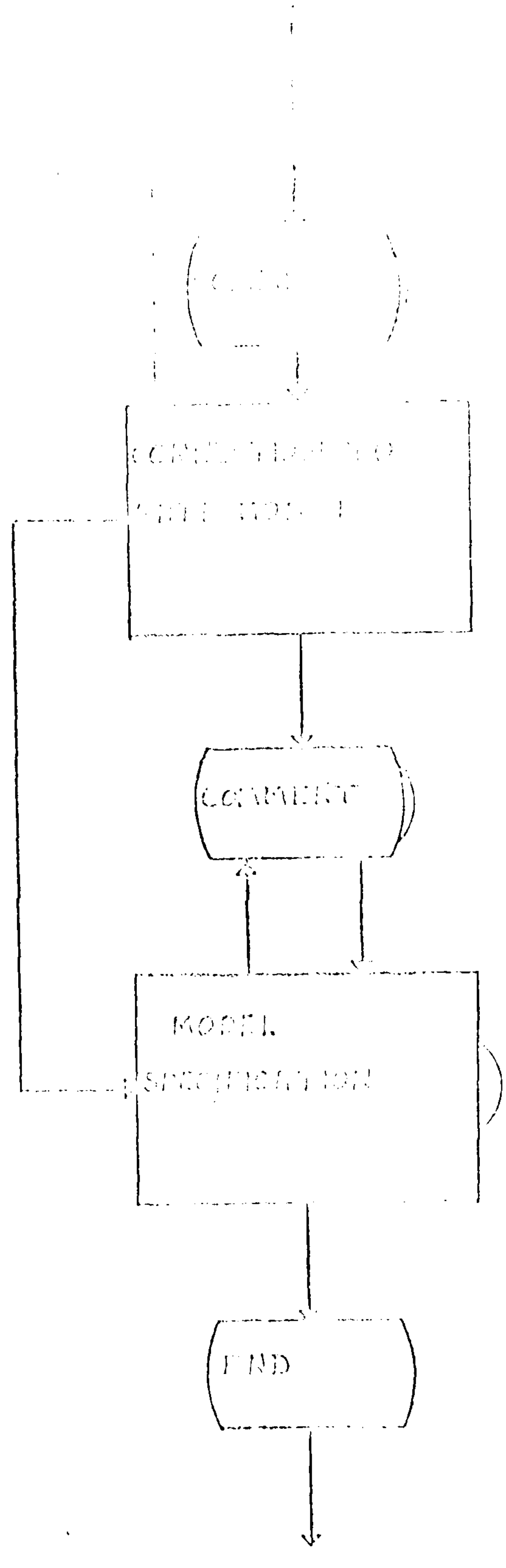




CONNECTION TO GATE MODEL




NULL


COMMENT
MODEL
SPECIFICATION
END

APPENDIX C

SISL LISTING

EACH SUBROUTINE IS PRECEDED BY ITS OWN OPERATING FLOW CHART.

THE FLOW CHARTS PORTRAY THE ALGORITHM USED FOR EACH ROUTINE.

In all the software, liberal use is made of certain functions which may not perform the same way on all computers:

OPEN, CLOSE are used for disk file handling.

.AND., .OR. are used on integer variables for data packing and and unpacking. These depend on a shift left being caused by I=I*2 and a shift right being caused by I=I/2 where I is an integer variable.

J="K places the non-decimal octal value K into the integer location I.

TABLE OF CONTENTS

```
C THIS FILE IS [illegible] DISK FILE, SISL.FOR
C
C
C SISL...S[illegible] THE SALOGS LANGUAGE
C
C [illegible]
C FOR THE GATE LEVEL SALOGS PORTION [illegible].
C
C THIS [illegible] A DIGITAL [illegible] DESCRIBED USING
C BEHAVIORAL [illegible] AND [illegible]
C [illegible] TO [illegible] OF THE TOTAL
C SYSTEM [illegible] IS
C SUBSEQUENT, [illegible] ORIGINAL
C VERSION OF IT WILL BE OVERWRITTEN BY SISL.
C THE BEHAVIORAL MODEL THUS GENERATED WILL INTERFACE TO THE
C SALOGS GATE LEVEL PORTION OF THE TOTAL SYSTEM DESCRIPTION.
C
C THIS PROGRAM...SISL.....SHOULD BE RUN
C BEFORE ...[illegible]...  THE FIRST PROGRAM IN THE SALOGS RUN TIME SYSTEM.
C
C AN ID (IDENTIFIER) IS A NODE NAME
C
C SYNTAX OF SISL MESSAGES-->
C   SUBROUTINE NAME, FORMAT NUMBER, MESSAGE
C
C THE "?" IN FNUM? IS THE "?" IN SUBROUTINE FNUM? WHICH IS THE
C BEHAVIOR MODEL WRITTEN IN FORTRAN IV TO DESCRIBE A
C BEHAVIORAL BLOCK. THE VARIOUS SUBROUTINES FNUM? UNIQUELY IDENTIFY A
C GIVEN BEHAVIORAL BLOCK TO SALOGS.
```

```
<[illegible] MAIN>
        V
        V
        V
        V
---------------------------------------------------------------
|
| C
| C [illegible]
| C
| C [illegible]
| C
| C
| C [illegible] THE FILES [illegible] BY SISL AT [illegible] TIME.
| C
|       CALL [illegible]
| C
| C [illegible] THE LIST OF ALLOWED [illegible] NAMES AND THEIR
| C [illegible] OF [illegible] NODES AND THE [illegible]?
| C
|       CALL [illegible]
| C
| C READ AND PROCESS THE [illegible] CARD. THIS SHOULD BE THE FIRST
| C SISL [illegible] CARD.
| C
|       CALL [illegible]
| C
| C TRANSLATE THE FUNCTIONAL SYSTEM DESCRIPTION INTO
| C [illegible] FUNCTIONAL AND LOGICAL MODEL SYNTAX
| C
|       CALL [illegible]
| C
| C [illegible] THE TRANSLATED INFORMATION FROM SISL.DAT ON TOP
| C OF SETUP.DAT
| C
| C
| C [illegible] TO "[illegible]" THE SALOGS FUNCTIONAL MODEL
| C [illegible].
| C
| C
| C [illegible] TO "[illegible]" THE SALOGS LOGICAL MODEL "CIRCUIT
| C
|       CALL [illegible]
| C
| C [illegible] "[illegible]" INTO "SETUP.DAT" AND DELETE "[illegible]"
| C
|       CALL [illegible]
| C
| C [illegible] ALL FILES [illegible] FOR THE SISL RUN TIME SYSTEM
| C [illegible] EXECUTION.
| C
|       CALL [illegible]
|--------------------------------------------------------------
        V
        V
        V
        V
---------------------------------------------------------------
|
|       [illegible]
|--------------------------------------------------------------
```

```
C
C
C
C DATA DESCRIPTION
C
C IBLANK [illegible] CHARACTER
C IASTRA  AN ASTERISK CHARACTER
C ICOLON  A SEMICOLON CHARACTER
C MAXNAM  THE MAXIMUM LENGTH OF A BLOCK NAME
C MAXNM1  MAXNAM+1
C MAXNM2  MAXNAM+2
C MAXNM3  MAXNAM+3
C MAXNM4  MAXNAM+4
C MAXBLK  THE MAXIMUM NUMBER OF BEHAVIORAL BLOCKS
C LSTCHR  A LIST OF CHARACTERS USED IN SISL
C NAMES   A LIST OF BLOCK NAMES AND THE CORRESPONDING
C         NUMBER OF OUTPUT/INPUT NODES REQUIRED BY EACH.
C NUMCHR  THE NUMBER OF CHARACTERS IN LSTCHR
C MAXLIN  THE LENGTH OF AN INPUT LINE
C ENDFIL  0, NO END OF FILE...1, END OF FILE REACHED (INPUT)
C IDTSIZ  THE SIZE OF IDTABL
C IDTABL  THE HASH TABLE FOR IDS
C MAXID   THE ALLOWED LENGTH OF AN IDENTIFIER
C MAXID1  MAXID+1
C MAXID2  MAXID+2
C NOID    0, ID FOUND ; 1, NO ID FOUND
C MAXCON  THE MAXIMUM NUMBER OF NODES WHICH CAN BE SHARED
C         BETWEEN THE FUNCTION AND GATE LEVEL PORTIONS
C         OF THE TOTAL SYSTEM DESCRIPTION.
C LINE    THE CURRENT WORKING INPUT LINE
C LINE1   A GENERAL ARRAY TO HOLD WORKING INFORMATION
C LINEND  THE END OF THE CURRENT WORKING INPUT LINE
C IDPNTR  THE BEGINNING OF THE NEXT IDENTIFIER IN THE
C         CURRENT WORKING INPUT LINE
C LSTCON  THE LIST OF OUTPUT AND INPUT NODE CONNECTIONS
C         THE SISL BEHAVIORAL MODEL SHARES WITH THE
C         SALOGS GATE MODEL
C NUMPUT  THE NUMBER OF NODE NAMES TO PUT ON A SINGLE LINE
C NUMCON  THE CURRENT NUMBER OF NODES IN LSTCON
C NUMIN   THE NUMBER OF INPUT NODES IN LSTCON
C NUMOUT  THE NUMBER OF OUTPUT NODES IN LSTCON
C MODCNT  THE NUMBER OF BEHAVIORAL MODELS REFERENCED
C         BY SISL.DAT
C
C
      COMMON/MODELS/ MODCNT,NUMPUT
C
      COMMON/IDS/ IDTABL(1000,10),IDPLAC,MAXID1,MAXID2,
     1 IDTSIZ
C
      COMMON/NAME/ NAMES(60,12),MAXNAM,MAXNM1,MAXNM2,MAXNM3,
     1 MAXNM4,MAXBLK,LSTCHR(70),NUMCHR
C
      COMMON/SISL/ LINE(80),LINE1(80),IBLANK,IASTRA,MAXLIN,
     1 LINEND,IDPNTR,MAXID,NOID,ENDFIL
C
      COMMON/NODES/ LSTCON(100,8),NUMCON,NUMOUT,NUMIN,MAXCON,
     1 ICOLON
```

```
C
C
C
      DATA NAMES,LSTCON/1520*1H /,LINE/80*0/,LINEND/0/
C
      DATA [illegible]/[illegible] /,[illegible],[illegible]D2/[illegible],9,10/
C
      DATA LSTCHR/ 1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,
     1 1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,1HW,
     2 1HX,1HY,1HZ,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,1H0,
     3 1H ,1H,,1H.,1H/,1H;,1H:,1H@,1H],1H_,1H-,1H[,1H^,1H<,
     4 1H>,1H?,1H+,1H*,1H=,1H),1H(,1H',1H&,1H%,1H$,1H#,1H",
     5 1H!,7*1H /
C
      DATA IBLANK,IASTRA,ICOLON,MAXLIN,ENDFIL,MAXID,MAXCON
     1 /1H ,1H*,1H;,80,0.0,8,100/
C
      DATA MODCNT,MAXNAM,MAXNM1,MAXNM2,MAXNM3,MAXNM4,MAXBLK,
     1 NUMCHR /0,8,9,10,11,12,60,70/
C
C OPEN ALL FILES
C
      CALL OPEN
C
C READ IN THE LIST OF ALLOWED BLOCK NAMES AND THEIR
C REQUIRED NUMBER OF OUTPUT/INPUT NODES AND THE FNUM?
C
      CALL READUM
C
C READ THE CONNECT CARD. THIS SHOULD BE THE FIRST
C SISL COMMAND CARD.
C
      CALL CONECT
C
C TRANSLATE THE BEHAVIORAL SYSTEM DESCRIPTION INTO
C SALOGS FUNCTIONAL AND LOGICAL MODEL SYNTAX
C
      CALL GETMOD
C
C PUT THE TRANSLATED INFORMATION FROM SISL.DAT ON TOP
C OF SETUP.DAT
C
C
C APPEND TO "TEMP1" THE SALOGS FUNCTIONAL MODEL
C INTERFACES.
C
C
C APPEND TO "TEMP1" THE SALOGS LOGICAL MODEL "CIRCUIT"
C
      CALL LMODEL
C
C PUT "TEMP1" INTO "SETUP.DAT" AND DELETE "TEMP1"
C
      CALL ENDMOD
C
C CLOSE ALL THE FILES AND TERMINATE EXECUTION.
C
      CALL CLOSE
      END
```

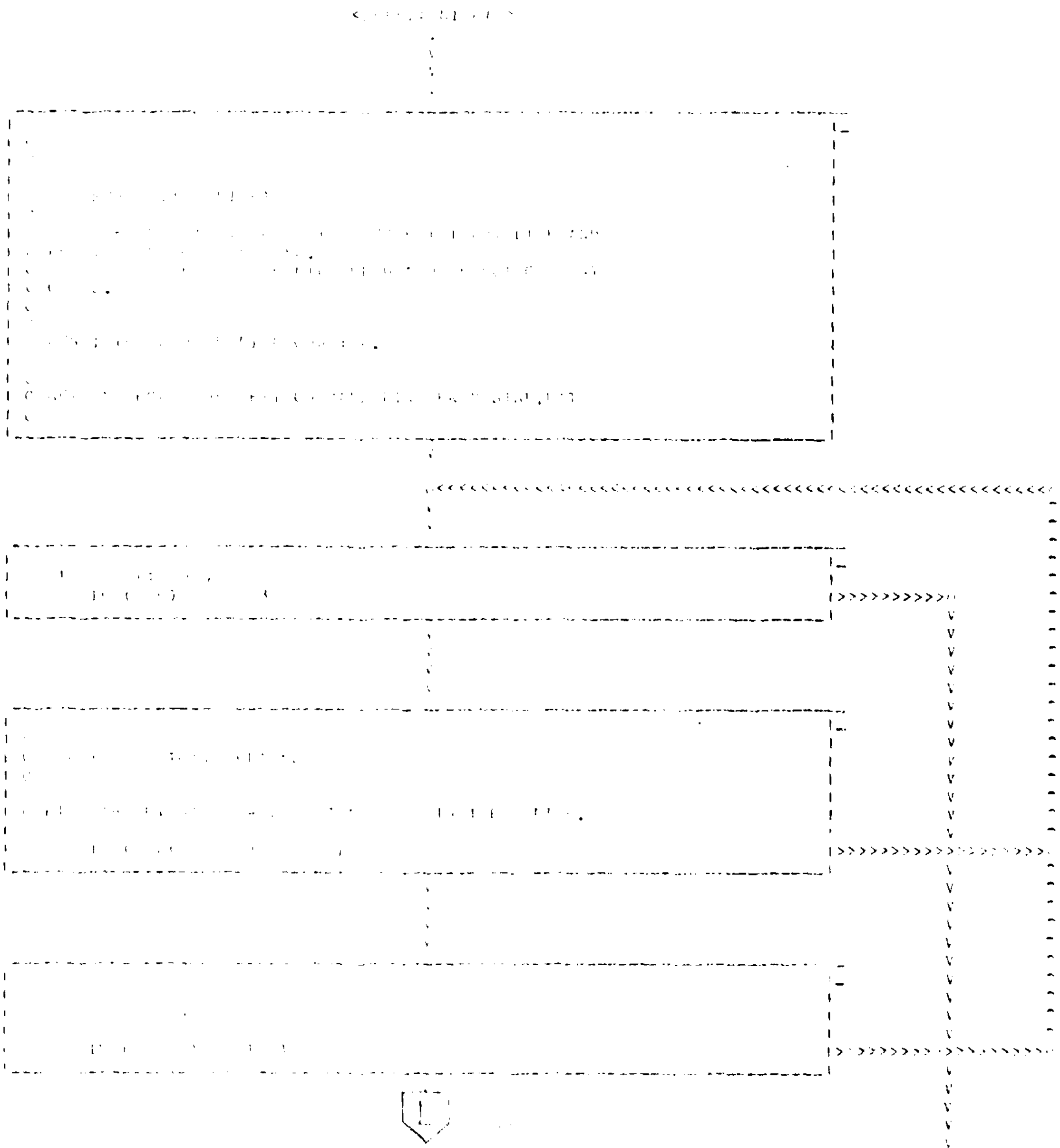
```
                         <[illegible]>
                               |
                               V

 -------------------------------------------------------------
| C
| C
|       END WHILE [illegible]
| C
| C [illegible] PROCEDURE.
| C
| C [illegible] FATAL ERROR MESSAGE GIVING
| C [illegible].
| C
| C CLOSE FILES AND TERMINATE EXECUTION
| C
|       CALL CLOSE
 -------------------------------------------------------------
                               V
                               V
                               V
                               V
 -------------------------------------------------------------
|       RETURN
 -------------------------------------------------------------

 -------------------------------------------------------------
|       END
 -------------------------------------------------------------
```

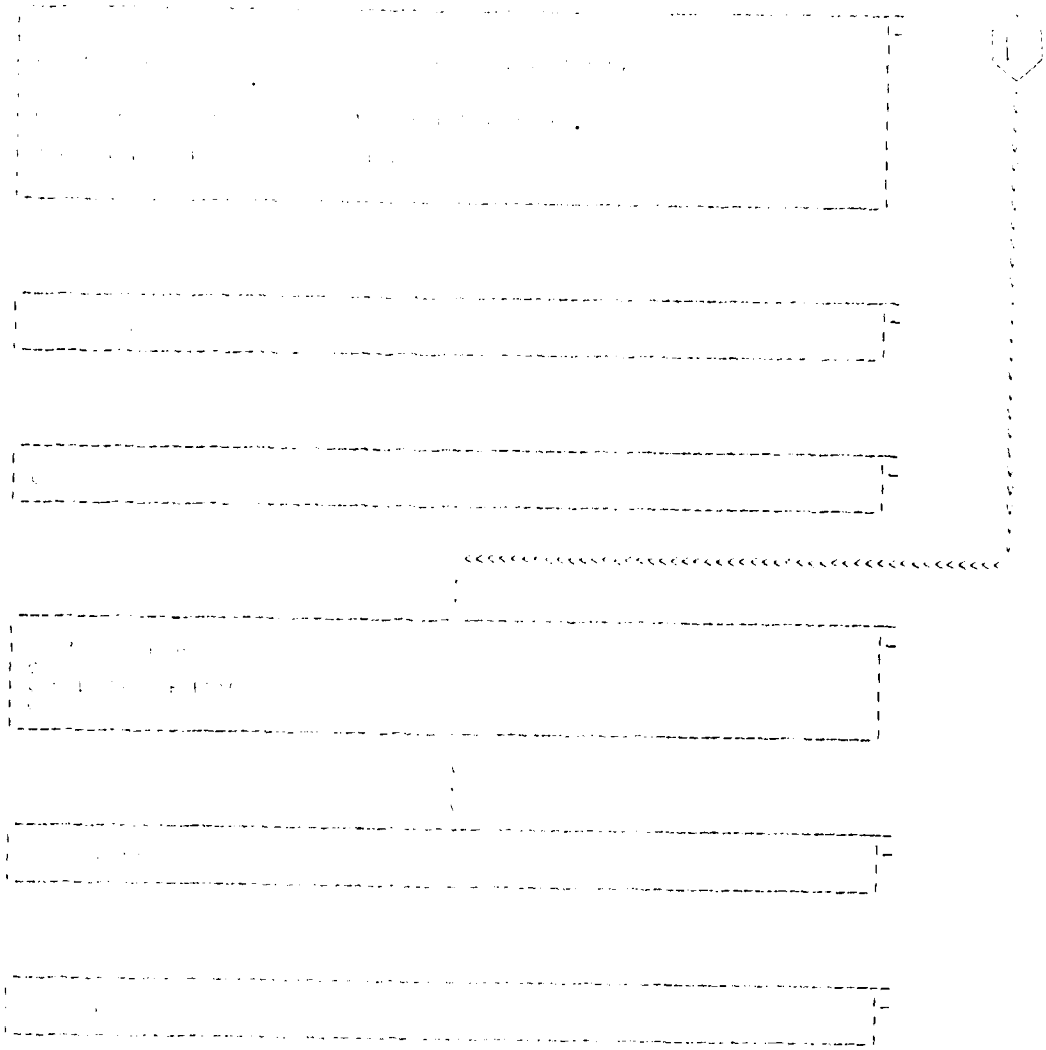
```
C
C
C
      SUBROUTINE HALT
C
C PROCESS THE FATAL ERROR THAT BEEN PRODUCED.
C
      COMMON/SISL/ LINE(80),LINE1(80),IBLANK,IASTRA,MAXLIN,
     1 LINEND,IDENTR,MAXID,NOID,ENDFIL
C
      WRITE (2,10)
      WRITE (5,10)
10    FORMAT (' HALT 10-- ** FATAL ERROR ** PROGRAM HALTED **')
      IF (LINEND.LE.0) GO TO 20
      WRITE (2,15) (LINE(I),I=1,LINEND)
      WRITE (5,15) (LINE(I),I=1,LINEND)
15    FORMAT (' HALT 15--',
     1        ' CURRENT SISL.DAT COMMAND LINE= ',80A1)
20    CALL CLOSE
      END
```

```
                         <Entry: [illegible] >
                                 |
                                 V

 -------------------------------------------------------------
| C                                                           |
| C                                                           |
| C                                                           |
|       SUBROUTINE [illegible]                                |
| C                                                           |
| C [illegible] ALL THE FILES [illegible] BY SISL AT [illegible] TIME. |
| C                                                           |
| C                                                           |
| C [illegible] THE FOLLOWING FILES:                          |
| C                                                           |
| C SISL.[illegible]    INPUT                                 |
| C [illegible]         INPUT                                 |
| C SISL.[illegible]    INPUT                                 |
| C [illegible]         OUTPUT                                |
| C SISL.[illegible]    OUTPUT                                |
| C [illegible]         OUTPUT                                |
| C                                                           |
 -------------------------------------------------------------
                                 V
                                 V
                                 V
                                 V
 -------------------------------------------------------------
|       RETURN                                                |
 -------------------------------------------------------------


 -------------------------------------------------------------
|       END                                                   |
 -------------------------------------------------------------
```

```
C
C
C
      SUBROUTINE OPEN
C
C OPENS ALL DATA FILES REQUIRED BY SISL AT RUN TIME.
C
      OPEN(UNIT=20,DEVICE='DSKC',ACCESS='SEQIN',MODE='ASCII',
     1 DISPOSE='SAVE',FILE='SISL.NAM')
      OPEN(UNIT=15,DEVICE='DSKC',ACCESS='SEQIN',MODE='ASCII',
     1 DISPOSE='SAVE',FILE='SETUP.DAT')
      OPEN(UNIT=10,DEVICE='DSKC',ACCESS='SEQIN',MODE='ASCII',
     1 DISPOSE='SAVE',FILE='SISL.DAT')
      OPEN(UNIT=3,DEVICE='DSKC',ACCESS='SEQOUT',MODE='ASCII',
     1 DISPOSE='SAVE',FILE='TEMP2')
      OPEN(UNIT=2,DEVICE='DSKC',ACCESS='SEQOUT',MODE='ASCII',
     1 DISPOSE='SAVE',FILE='SISL.OUT')
      OPEN(UNIT=1,DEVICE='DSKC',ACCESS='SEQOUT',MODE='ASCII',
     1 DISPOSE='SAVE',FILE='TEMP1')
      REWIND 20
      REWIND 15
      REWIND 10
      REWIND 3
      REWIND 2
      REWIND 1
      RETURN
      END
```

```
C
C
C
      SUBROUTINE CLOSE
C
C CL[illegible] FOR [illegible] OF [illegible] AND
C TERMINATES EXECUTION.
C
      CLOSE (UNIT=20)
      CLOSE (UNIT=15)
      CLOSE (UNIT=10)
      CLOSE (UNIT=3)
      CLOSE (UNIT=2)
      CLOSE (UNIT=1)
      STOP
      END
```

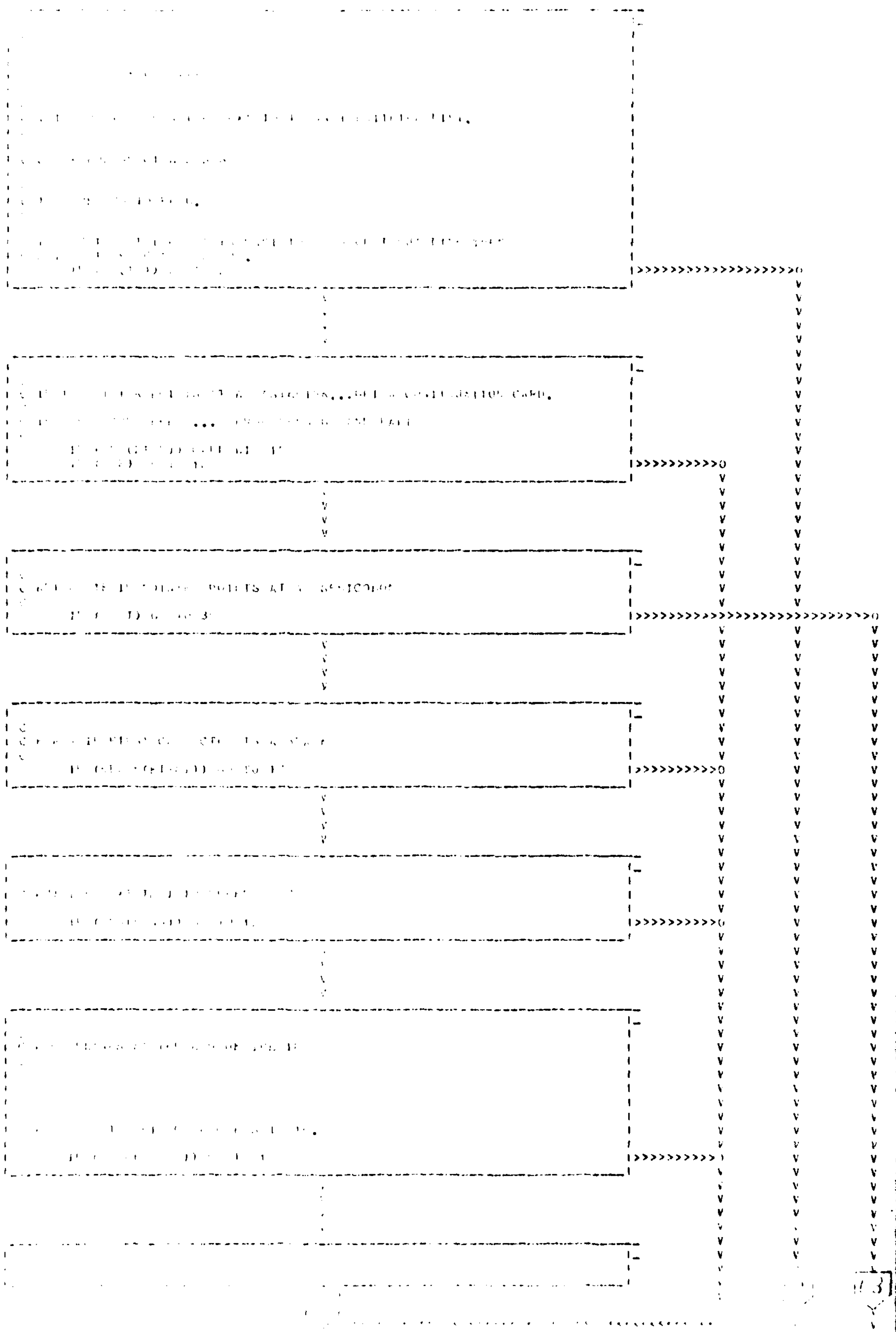




```
C
C
C
      SUBROUTINE RLINE
C
C [illegible]
C [illegible]
C [illegible] FLAG, REDUCE [illegible]
C CARD, AND [illegible]
C
      COMMON/SIGH/ LINE(80),LINE1(80),IBLANK,IASTRA,MANLIN,
     1 LINEND,IDPNTR,[illegible]
C
C
C GET THE NEXT COMMAND LINE FROM SIGH.DAT
C
      LINEND 0
5     READ (10,10,END=1000) LINE1
10    FORMAT (80A1)
      WRITE (2,15) LINE1
      WRITE (5,15) LINE1
15    FORMAT (' LINEIN 15-- ',80A1)
C
C DELETE DUPLICATE BLANKS
C
      DO 20 I=1,MANLIN
         IF (LINE1(I).NE.IBLANK) GO TO 25
20    CONTINUE
      GO TO 5
25    IF (LINE1(I).EQ.IASTRA) GO TO 5
27    LINEND=LINEND+1
      LINE(LINEND)=LINE1(I)
      I=I+1
      IF (I.GT.MANLIN) GO TO 500
      IF (LINE1(I).NE.IBLANK) GO TO 27
      LINEND=LINEND+1
      LINE(LINEND)=IBLANK
      K=I+1
      IF (K.GT.MANLIN) GO TO 500
      DO 30 I=K,MANLIN
         IF (LINE1(I).NE.IBLANK) GO TO 27
30    CONTINUE
C
C RESET THE LINE POSITION POINTER
C
500   IDPNTR 1
      I=LINEND+1
      IF (I.GT.MANLIN) GO TO 600
      DO 550 J=I,MANLIN
         LINE(J)=IBLANK
550   CONTINUE
600   RETURN
C
C SET THE [illegible] FLAG
C
1000  [illegible] 1.0
      RETURN
      END
```

AD-A100 784 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
A FUNCTIONAL LEVEL PREPROCESSOR FOR COMPUTER AIDED DIGITAL DESI--ETC(U)
DEC 80 P G RAETH
UNCLASSIFIED AFIT/GCS/EE/80D-12 NL

2 of 3
AD
A100 784



```
C
C
C
      SUBROUTINE NEXTID
C
      COMMON/IDS/ IDTABL([illegible],10),IDPLAC,MAXID1,MAXID2,
     1 IDTSIZ
      COMMON/SISL/ LINE(80),LINE1(80),IBLANK,IASTRA,MAXLIN,
     1 LINEND,IDPNTR,MAXID,NOID,ENDFIL
      COMMON/NODES/ LSTCON(100,8),NUMCON,NUMOUT,NUMIN,MAXCON,
     1 ICOLON
C
C THIS ROUTINE GETS THE NEXT ID IN AN IDENTIFIER LIST.
C
C
C CHECK FOR SPECIAL CASES
C
      NOID=0
      IF (IDPNTR.GT.LINEND) GO TO 500
      IF (LINE(IDPNTR).EQ.IASTRA) CALL LINEIN
      IF (ENDFIL.EQ.1.0) GO TO 550
      IF (LINE(IDPNTR).EQ.ICOLON) GO TO 475
      IF (LINE(IDPNTR).EQ.IBLANK) GO TO 1500
C
C PUT THE NEXT ID INTO LINE1
C
      DO 10 I=1,MAXID
         IF (LINE(IDPNTR).EQ.IBLANK) GO TO 100
         LINE1(I)=LINE(IDPNTR)
         IDPNTR=IDPNTR+1
         IF (IDPNTR.GT.LINEND) GO TO 100
10    CONTINUE
C
C PACK BLANKS AT THE END OF THE ID
C
      I=MAXID+1
100   IF (IDPNTR.GT.LINEND) I=I+1
      IF (LINE(IDPNTR).NE.IBLANK) GO TO 600
      IF (I.GT.MAXID) GO TO 200
      DO 150 K=I,MAXID
         LINE1(K)=IBLANK
150   CONTINUE
200   IDPNTR=IDPNTR+1
      DO 210 I=1,MAXID
         IF (LINE1(I).EQ.IBLANK) GO TO 400
         ICHR=LINE1(I)
         IF (ICHAR(ICHR).GT.36) GO TO 700
210   CONTINUE
400   IDPLAC=IDHASH(IDUMMY)
475   RETURN
C
C NO ID FOUND
C
500   NOID=1
      RETURN
```

```
C
C ERROR MESSAGES
C
550   WRITE (2,551)
      WRITE (5,551)
551   FORMAT (' NEXID 551-- EOF WHILE TRYING TO GET',
     1        ' CONTINUATION CARD')
      CALL HALT
600   WRITE (2,601)
      WRITE (5,601)
601   FORMAT (' NEXID 601-- ID TOO LONG')
      CALL HALT
700   WRITE (2,701) (LINE1(I),I=1,MAXID)
      WRITE (5,701) (LINE1(I),I=1,MAXID)
701   FORMAT (' NEXID 701-- ID CONTAINS INVALID CHARACTERS',
     1        ' (A-Z,0-9 ONLY) ',10A1)
      CALL HALT
1500  WRITE (2,1501) IDPNTR
      WRITE (5,1501) IDPNTR
1501  FORMAT (' NEXTID 1501-- A BLANK IS THE FIRST CHARACTER',
     1        ' OF AN IDENTIFIER',/,' IDPNTR= ',I10)
      CALL HALT
      END
```

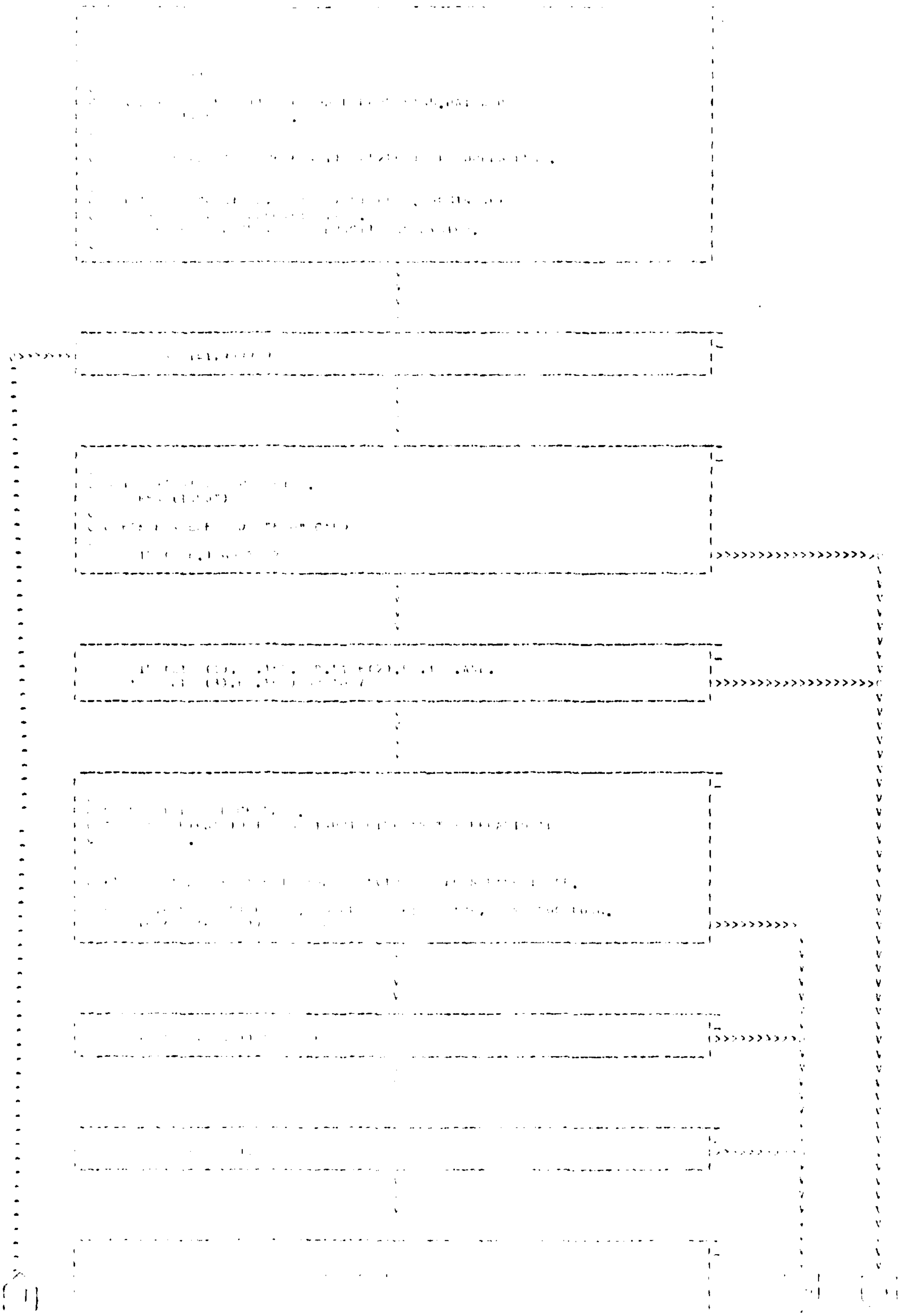
```
                    <ENTRY: LMODEL>
                           V
                           V
                           V
                           V
 -----------------------------------------------------------------
 |                                                               |
 | C                                                             |
 | C                                                             |
 | C                                                             |
 |       SUBROUTINE LMODEL                                       |
 | C                                                             |
 | C THIS ROUTINE CREATES THE SALOGS LOGICAL MODEL "CIRCUIT"     |
 | C AND APPENDS IT TO THE FUNCTIONAL MODEL LIST NOW RESIDING    |
 | C IN FILE "TAPE1"                                             |
 | C                                                             |
 | C                                                             |
 | C PRINT THE "CIRCUIT" LIST                                    |
 | C THIS IS THE BEGINNING OF THE SALOGS LOGICAL MODEL WHICH     |
 | C DESCRIBES THE OVERALL MACRO SYSTEM.                         |
 | C                                                             |
 | C                                                             |
 | C STORE THE APPROPRIATE SALOGS LOGICAL MODEL NAME WITH ITS    |
 | C REQUIRED PARAMETERS.                                        |
 | C                                                             |
 | C                                                             |
 | C PRINT THE OUTPUT/INPUT LIST                                 |
 | C                                                             |
 | C                                                             |
 | C APPEND THE SISL SYSTEM BLOCK NAMES WITH THEIR PARAMETERS    |
 | C                                                             |
 | C                                                             |
 | C APPEND THE SALOGS LOGICAL MODEL END FLAG.                   |
 | C                                                             |
 | C APPEND THE SALOGS END OF FUNCTIONAL/LOGICAL MODELS END FLAG.|
 | C                                                             |
 -----------------------------------------------------------------
                           V
                           V
                           V
                           V
 -----------------------------------------------------------------
 |                                                               |
 |       RETURN                                                  |
 |                                                               |
 -----------------------------------------------------------------


 -----------------------------------------------------------------
 |                                                               |
 |       END                                                     |
 |                                                               |
 -----------------------------------------------------------------
```

```
C
C
C
      SUBROUTINE LMODEL
C
      COMMON/MODELS/ MODCNT,NUMPUT
      COMMON/NODES/ LSTCON(100,8),NUMCON,NUMOUT,NUMIN,MAXCON,
     1 ICOLON
      COMMON/SISL/ LINE(80),LINE1(80),IBLANK,IASTRA,MAXLIN,
     1 LINEND,IDPNTR,MAXID,NOID,ENDFIL
C
C THIS ROUTINE CREATES THE SALOGS LOGICAL MODEL "CIRCUIT"
C AND APPENDS IT TO THE BEHAVIORAL MODEL LIST NOW RESIDING
C IN FILE "TEMP1"
C
C
C PRINT THE "CIRCUIT" LINE
C
      WRITE (2,5)
      WRITE (5,5)
5     FORMAT(' LMODEL 5-- AM BUILDING THE SALOGS LOGICAL MODEL')
      WRITE (1,10) MODCNT,NUMOUT,NUMIN,NUMCON
10    FORMAT ('CIRCUIT',4(1X,I4),'    0    0')
      NUMPUT=80/(MAXID+5)
      IF (NUMPUT.LT.1) GO TO 500
      M=0
      N=1-NUMPUT
C
C PRINT THE OUTPUT/INPUT LIST
C
15    M=M+NUMPUT
      N=N+NUMPUT
      IF (M.GT.NUMCON) M=NUMCON
      IF (N.GT.M) GO TO 50
      IPLACE=0
      DO 20 J=N,M
         DO 18 K=1,MAXID
            IPLACE=IPLACE+1
            LINE1(IPLACE)=LSTCON(J,K)
18       CONTINUE
         IPLACE=IPLACE+1
         LINE1(IPLACE)=IBLANK
20    CONTINUE
      IF (NUMCON.GT.M) IPLACE=IPLACE+1
      IF (NUMCON.GT.M) LINE1(IPLACE)=IASTRA
      WRITE (1,25) (LINE1(I),I=1,IPLACE)
25    FORMAT (80A1)
      IF (NUMCON.GT.M) GO TO 15
50    IF (MODCNT.LT.1) GO TO 70
```

```
C
C LIST THE BEHAVIORAL MODEL LINES
C THESE RESIDE IN TEMP2
C
      CLOSE (UNIT=3)
      OPEN(UNIT=3,DEVICE='DSK',ACCESS='SEQIN',MODE='ASCII',
     1 DISPOSE='DELETE',FILE='TEMP2')
      REWIND 3
55    READ (3,25,END=70) LINE1
      WRITE (1,25) LINE1
      GO TO 55
70    WRITE (1,71)
71    FORMAT ('END CIRCUIT',/,'$END MODELS')
      RETURN
C
C ERROR MESSAGES
C
500   WRITE (2,501)
      WRITE (5,501)
501   FORMAT (' LMODEL 501-- NUMPUT<1, MUST BE>1',/,
     1        ' THIS IS CAUSED BY NODE NAMES BEING TOO BIG',/,
     2        ' DECREASE THE ALLOWED NODE NAME LENGTH')
      CALL HALT
      END
```

```
                         <Entry: [illegible]>
                                 V
                                 V
                                 V
                                 V
 ---------------------------------------------------------------------
 | C                                                                 |
 | C                                                                 |
 | C                                                                 |
 |       SUBROUTINE [illegible]                                      |
 | C                                                                 |
 |       COMMON /SISL/ LINE([illegible]),LINE1([illegible]),IPLANE,IASIPA,MAXLIN,
 |      + [illegible]                                                |
 | C                                                                 |
 | C REVISE SETUP.DAT TO HAVE AT ITS HEAD THE NEW MODELING           |
 | C INFORMATION CREATED BY SISL.EXE                                 |
 | C                                                                 |
 | C                                                                 |
 | C GET "SETUP.DAT" AND APPEND IT TO "TEMP1"                        |
 | C                                                                 |
 | C                                                                 |
 | C APPEND THE SALOGS GATE LEVEL PORTION OF THE DIGITAL             |
 | C SYSTEM TO THE FUNCTIONAL/LOGICAL MODELS CREATED IN              |
 | C SUBROUTINE [illegible].                                         |
 | C                                                                 |
 | C                                                                 |
 | C TRANSFER "TEMP1" TO "SETUP.DAT"                                 |
 | C                                                                 |
 | C                                                                 |
 | C SPOOL TOTAL SYSTEM DESCRIPTION TO THE FILE TO BE PASSED ON      |
 | C TO SALOGS.                                                      |
 | C                                                                 |
 ---------------------------------------------------------------------
                                 V
                                 V
                                 V
                                 V
 ---------------------------------------------------------------------
 |                                                                   |
 |       RETURN                                                      |
 |                                                                   |
 ---------------------------------------------------------------------


 ---------------------------------------------------------------------
 |                                                                   |
 |       END                                                         |
 |                                                                   |
 ---------------------------------------------------------------------
```

```
C
C
C
      SUBROUTINE ENDMOD
C
      COMMON/SISL/ LINE(80),LINE1(80),IBLANK,IASTRA,MAXLIN,
     1 LINEND,IDPNTR,MAXID,NOLD,ENDFIL
C
C REMAKE SETUP.DAT TO HAVE AT ITS HEAD THE NEW MODELING
C INFORMATION CREATED BY SISL.EXE
C
C
C GET "SETUP.DAT" AND APPEND IT TO "TEMP1"
C
      WRITE (2,3)
      WRITE (5,3)
3     FORMAT (' ENDMOD 3-- AM REBUILDING SETUP.DAT FOR SALOGS')
1     READ (15,5,END=100) LINE1
      WRITE (1,5) LINE1
5     FORMAT (80A1)
      GO TO 1
C
C TRANSFER "TEMP1" TO "SETUP.DAT"
C
100   CLOSE(UNIT=15)
      CLOSE(UNIT=1)
      OPEN(UNIT=15,DEVICE='DSKC',ACCESS='SEQOUT',MODE='ASCII',
     1 DISPOSE='SAVE',FILE='SETUP.DAT')
      OPEN(UNIT=1,DEVICE='DSKC',ACCESS='SEQIN',MODE='ASCII',
     1 DISPOSE='DELETE',FILE='TEMP1')
      REWIND 15
      REWIND 1
110   READ (1,5,END=200) LINE1
      WRITE (15,5) LINE1
      GO TO 110
200   RETURN
      END
```



[illegible]

[illegible]

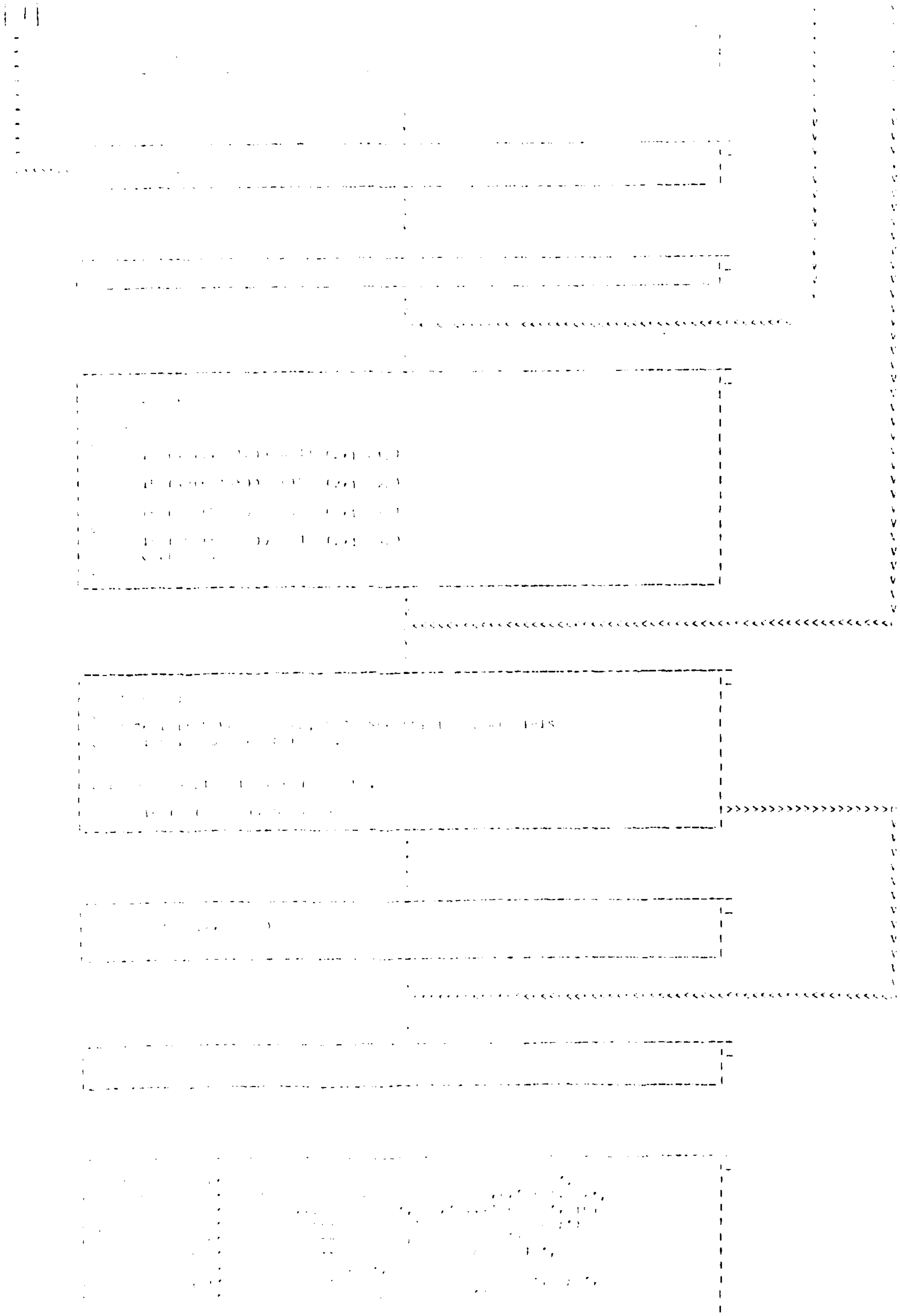
[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



```
C
C
C
      SUBROUTINE GETMOD
C
      COMMON/[illegible]/ [illegible]([illegible],10),[illegible],MAXID1,MAXID2,
     1 IDTSIZ
      COMMON/[illegible]/ [illegible]
      COMMON/[illegible]/ LINE([illegible]),LINE1(30),IBLANK,IASTRA,MAXLIN,
     1 LINEND,IDPNTR,[illegible],[illegible],ENDFIL
      COMMON/NAME/ NAMES(60,12),MAXNAM,MAXNM1,MAXNM2,MAXNM3,
     1 MAXNM4,LABLE,[illegible](70),[illegible]
      COMMON/NODES/ LSTCON(100,3),NUMCON,NUMOUT,NUMIN,MAXCON,
     1 ICOLON
C
C READS IN THE BEHAVIORAL MODEL FROM SISL.DAT AND
C ARRANGES IT FOR SALOGS.
C
      WRITE (2,1)
      WRITE (5,1)
1     FORMAT (' GETMOD 2-- AM BUILDING THE SALOGS FUNCTIONAL',
     1        ' MODELS')
C
C GET THE NEXT BLOCK IN THE BEHAVIORAL SYSTEM
C
      WRITE (1,3)
3     FORMAT ('$MODELS')
5     NOUT=0
      NTOTAL=0
      OUTFLG=1.
      CALL LINEIN
C
C CHECK FOR EOF
C
      IF (ENDFIL.EQ.1.) GO TO 1000
      IF (LINE(1).EQ.1HE.AND.LINE(2).EQ.1HN.AND.
     1    LINE(3).EQ.1HD) GO TO 1000
C
C PULL OUT THE BLOCK NAME
C
      MODCNT=MODCNT+1
      DO 10 NUMHSH=1,MAXNAM
         LINE1(NUMHSH)=LINE(NUMHSH)
         IF (LINE(NUMHSH).EQ.IBLANK) GO TO 20
10    CONTINUE
      IF (LINE(MAXNM).NE.IBLANK) GO TO 600
      NUMHSH=MAXNM1
20    NUMHSH=NUMHSH-1
      IPLACE=IFIND(NUMHSH)
      IF (IPLACE.EQ.0) GO TO 500
      IDPNTR=NUMHSH+2
      WRITE (1,22) (NAMES(IPLACE,I),I=1,MAXNAM),
     1 (NAMES(IPLACE,I),I=MAXNM1,MAXNM4)
22    FORMAT (8A1,'    0',4I5,'    0')
      IDPNT1=IDPNTR
      IF (LINE(IDPNTR).EQ.IASTRA) WRITE (3,105) LINE
      IF (LINE(IDPNTR).EQ.IASTRA) IDPNT1=1
```

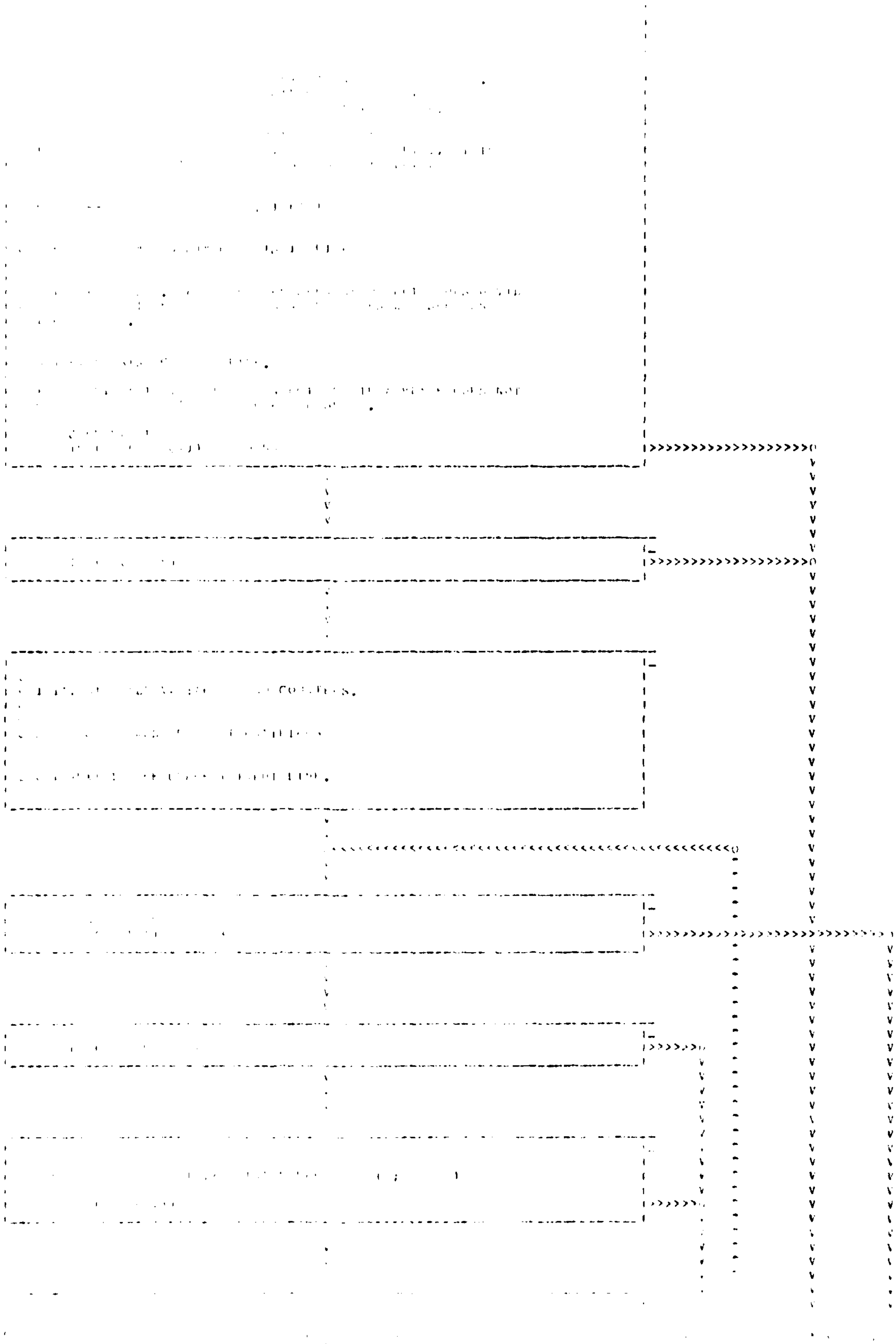
```
C
C CHECK ALL THE IDS OF THE GIVEN BLOCK, REMOVE THE ;
C
30    CALL NEXTID
      IF (NOID.EQ.1) GO TO 120
C
C CASE FOR SEMICOLON
C
      IF (LINE(IDPNTR).NE.ICOLON) GO TO 100
      IF (LINE(IDPNTR+1).NE.IBLANK) GO TO 700
      LINE(IDPNTR)=IBLANK
      OUTFLG=0.
      IDPNTR=IDPNTR+2
      IF (IDPNTR.EQ.3) NOUT=NTOTAL
      IF (IDPNTR.EQ.3) GO TO 30
      NOUT=NTOTAL+1
      IDTABL(IDPLAC,MAXID1)=1
C
C CASE FOR ASTERISK
C
100   IF (LINE(IDPNTR).NE.IASTRA) GO TO 110
      WRITE (3,105) LINE
105   FORMAT (80A1)
      WRITE (1,105) (LINE(I),I=IDPNT1,MAXLIN)
      IDPNT1=1
C
C CASE FOR VALID ID
C
110   NTOTAL=NTOTAL+1
      IF (OUTFLG.EQ.1.) IDTABL(IDPLAC,MAXID1)=1
      IF (OUTFLG.EQ.0.) IDTABL(IDPLAC,MAXID2)=1
      GO TO 30
120   IF (NOUT.EQ.0) NINPUT=NTOTAL
      IF (NOUT.NE.0) NINPUT=NTOTAL-NOUT
      WRITE (3,105) LINE
      WRITE (1,105) (LINE(I),I=IDPNT1,MAXLIN)
      WRITE (1,123) (NAMES(IPLACE,I),I=1,MAXNAM)
123   FORMAT (4HEND ,8A1)
      IF (NOUT.EQ.NAMES(IPLACE,MAXNM1).AND.
     1    NINPUT.EQ.NAMES(IPLACE,MAXNM2)) GO TO 5
```

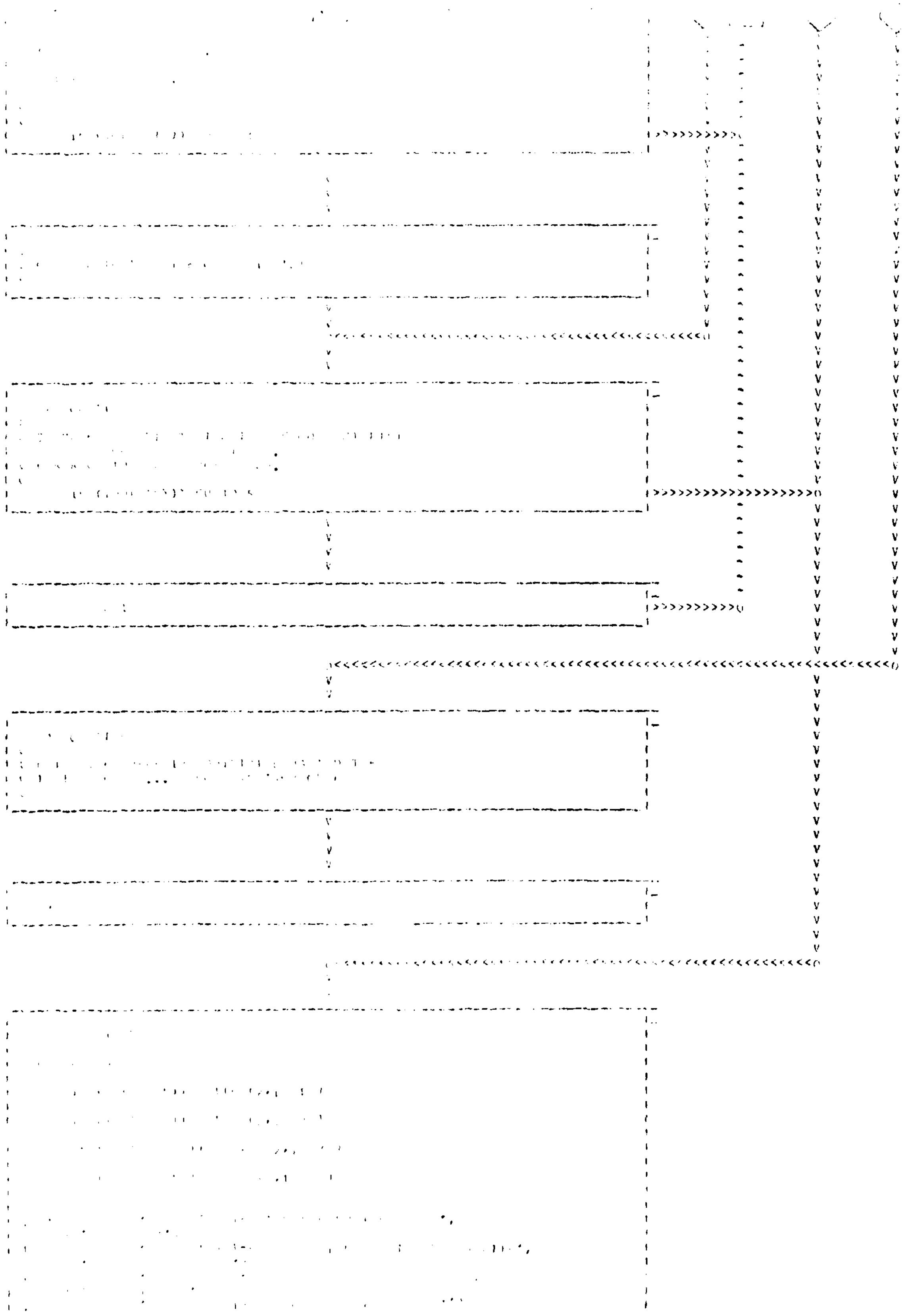
```
C
C ERROR MESSAGES
C
      WRITE (2,125) NOUT,NAMES(IPLACE,MAXSH1),
     1              NINPUT,NAMES(IPLACE,MAXSH2)
      WRITE (5,125) NOUT,NAMES(IPLACE,MAXSH1),
     1              NINPUT,NAMES(IPLACE,MAXSH2)
125   FORMAT (' GETMOD 125-- INVALID NUMBER OF OUTPUT',
     1        ' OR INPUT LINES FOR THIS BLOCK',/,' (GIVEN/',
     2        'REQUIRED....OUTPUT= ',2I5,' ...INPUT= ',2I5)
      CALL HALT
500   WRITE (2,501) (LINE1(I),I=1,NUMHSH)
      WRITE (5,501) (LINE1(I),I=1,NUMHSH)
501   FORMAT (' GETMOD 501-- BLOCK NAME NOT FOUND IN',
     1        ' NAMES TABLE...= ',10A1)
      CALL HALT
600   WRITE (2,601)
      WRITE (5,601)
601   FORMAT (' GETMOD 601-- IDENTIFIER TOO LONG')
      CALL HALT
700   WRITE (2,701)
      WRITE (5,701)
701   FORMAT (' GETMOD 701-- A BLANK ALWAYS FOLLOWS A ;')
      CALL HALT
```

```
C
C CHECK IDTABL FOR ERRORS, SUCCESSFULLY TERMINATE THIS
C ROUTINE IF NO ERRORS FOUND.
C
1000  ERRFLG=0.
      WRITE (2,1002)
      WRITE (5,1002)
1002  FORMAT (/,' GETMOD 1002- ',/,' NODE NAME',5X,
     1          'OUTFLAG',5X,'INFLAG',5X,'HASH #')
      DO 1010 I=1,IDTSIZ
         IF (IDTABL(I,1).EQ.IBLANK) GO TO 1010
         WRITE (2,1003) (IDTABL(I,J),J=1,MAXID2),I
         WRITE (5,1003) (IDTABL(I,J),J=1,MAXID2),I
1003     FORMAT (T2,8A1,8X,I5,6X,I5,6X,I5)
         IF (IDTABL(I,MAXID1).EQ.IDTABL(I,MAXID2)) GO TO 1010
         WRITE (2,1005)
         WRITE (5,1005)
1005     FORMAT (' GETMOD 1005-- THE ABOVE ID DOESNT HAVE AN',
     1          ' INPUT AND OUTPUT CONNECTION ')
         ERRFLG=1.
1010  CONTINUE
      WRITE (2,1020)
      WRITE (5,1020)
1020  FORMAT (1X)
      IF (ERRFLG.EQ.1.) CALL HALT
      RETURN
      END
```

```
<[illegible]>
      v
      v
-------------------------------------------------------
| C
| C
| C
|      [illegible]
| C
| C [illegible]
|
| C [illegible].
| C
| C [illegible].
-------------------------------------------------------
      v
      v
      0<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<0
      v                                             .
      v                                             .
-------------------------------------------------------
|   1 CONTINUE                                      |
| C                                                 |
|      IF ([illegible]) GO TO 2                     |>>>>>>>>>>0
-------------------------------------------------------          v
      v                                                          v
      v                                                          v
-------------------------------------------------------          v
| C                                                 |            v
| C [illegible]                                     |            v
| C                                                 |            v
| C [illegible].                                    |            v
| C                                                 |            v
| C [illegible].                                    |            v
|                                                   |            v
|      [illegible]                                  |>>>>>>>>>>>>>>>>>>0
-------------------------------------------------------          v
                                                                 v
      0<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
      v
      v
-------------------------------------------------------
|   2 [illegible]                                   |
-------------------------------------------------------

-------------------------------------------------------
|      [illegible]                                  |
-------------------------------------------------------
```

```
C
C
C
      SUB[illegible] P[illegible]ID
C
      COM[illegible] / [illegible],MA[illegible]
     1 [illegible],[illegible],[illegible]
      COM[illegible] / [illegible](100,8),[illegible]ROW,NEWOUT,NUMIN,MAXCOR,
     1 [illegible]
C
C THIS ROUTINE PROVIDES A CHECK ON PICKING UP IDENTIFIERS.
C IT IS USED ONLY IF THE PERSON MAINTAINING SISI WISHES TO HAVE
C A TEST OF ID RETRIEVAL.
C
      CALL LINEIN
5     CALL NEXTID
      IF (NO[illegible].EQ.1) GO TO 100
      IF (LINE(IDPNTR).EQ.ICOLON) IDPNTR=IDPNTR+2
      WRITE (2,10) (LINE1(I),I=1,MAXID)
      WRITE (5,10) (LINE1(I),I=1,MAXID)
10    FORMAT (T2,10A1)
      GO TO 5
100   WRITE (5,101)
101   FORMAT (' NO MORE IDENTIFIERS')
      RETURN
      END
```

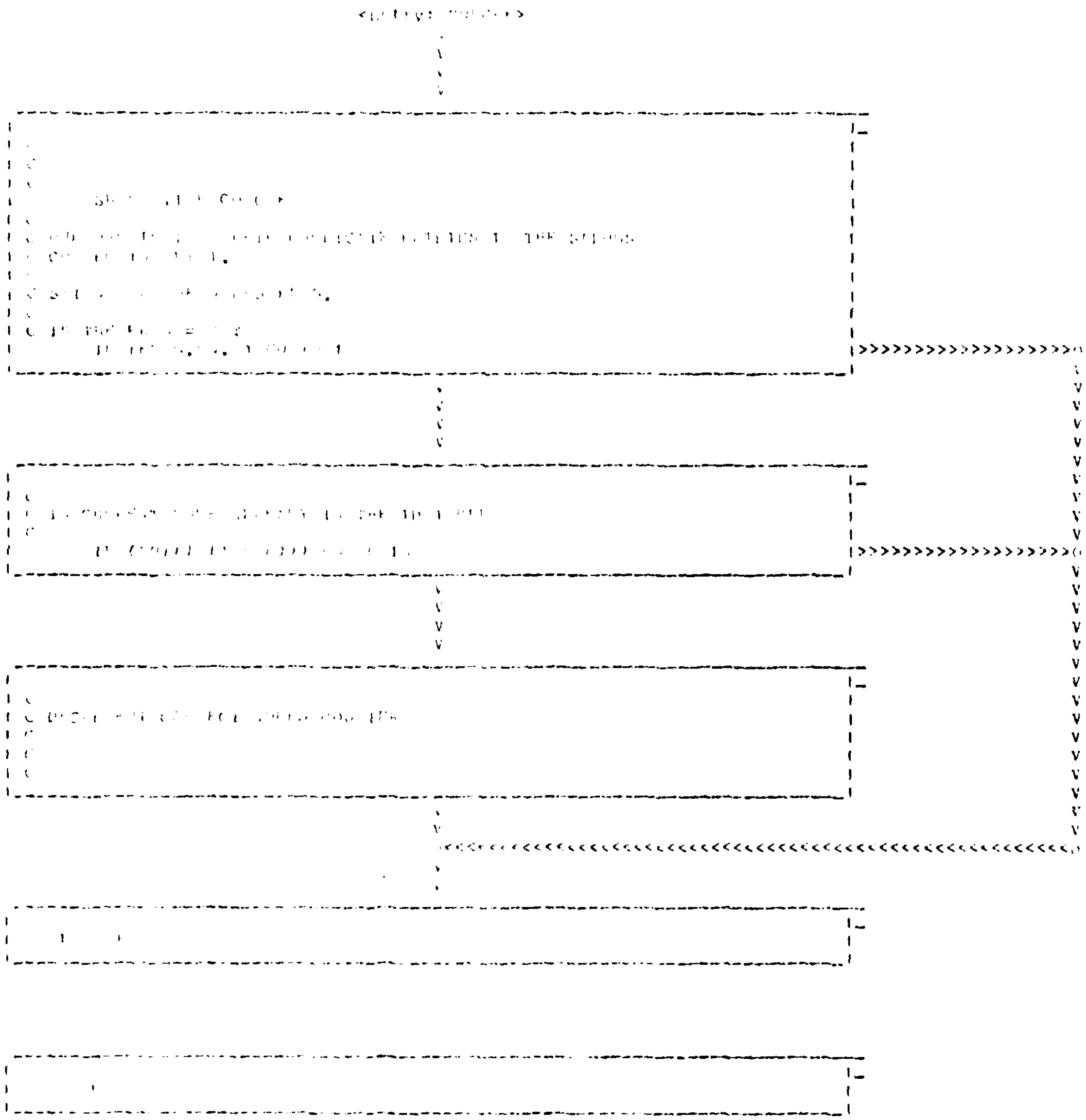




```
C
C
C
      SUBROUTINE CONECT
C
      COMMON/IDS/ IDTABL(1000,10),IDPLAC,MAXID1,MAXID2,
     1 IDTSIZ
      COMMON/SISL/ LINE(80),LINEI(80),IBLANK,IASTRA,MAXLIN,
     1 LINEND,IDPNTR,MAXID,NOID,ENDFIL
      COMMON/NODES/ LSTCON(100,8),NUMCON,NUMOUT,NUMIN,MAXCON,
     1 ICOLON
C
C PURPOSE IS TO OPERATE ON THE FIRST SISL PROGRAM CARD.
C THIS CARD SHOULD BE A "CONNECT" CARD WHICH LISTS ALL
C THE NODES COMMON TO THE SALOGS GATE LEVEL MODEL.
C
C IN ONE SENSE  OUTPUT MEANS OUTPUT FROM A BLOCK
C IF A NODE IS NOTED AS BEING IN THE CONNECT LIST, IT IS
C REFERED TO AS AN INPUT TO THE SALOGS GATE LEVEL MODEL
C
C THIS ROUTINE WORKS MUCH THE SAME AS GETMOD AND WAS USED AS A
C MODEL FOR CREATING IT.
C
C FORMAT--> CONNECT OUTLIST ; INLIST
C
C
C MAKE SURE A "CONNECT" CARD IS THE FIRST SISL COMMAND CARD.
C
      CALL LINEIN
      IF (LINE(1).NE.1HC.OR.LINE(2).NE.1HO.OR.
     1    LINE(3).NE.1HN.OR.LINE(4).NE.1HN.OR.
     2    LINE(5).NE.1HE.OR.LINE(6).NE.1HC.OR.
     3    LINE(7).NE.1HT) GO TO 500
      IF (LINE(8).NE.IBLANK) GO TO 600
      OUTFLG=1.
      NUMOUT=0
      NUMCON=0
      IDPNTR=9
C
C LOOP TO GET ALL OF THE IDENTIFIERS
C
10    CALL NEXTID
      IF (NOID.EQ.1) GO TO 400
      IF (IDPNTR.GT.LINEND) GO TO 15
C
C CHECK FOR THE OUTLIST/INLIST SEPARATOR  (;)
C
      IF (LINE(IDPNTR).NE.ICOLON) GO TO 15
      IF (LINE(IDPNTR+1).NE.IBLANK) GO TO 800
      OUTFLG=0.
      IDPNTR=IDPNTR+2
      IF (IDPNTR.EQ.3) NUMOUT=NUMCON
      IF (IDPNTR.NE.3.AND.NUMCON.NE.0) IDTABL(IDPLAC,MAXID2)=1
      IF (IDPNTR.NE.3.AND.NUMCON.NE.0) NUMOUT=NUMCON+1
      IF (IDPNTR.EQ.3.OR.NUMCON.EQ.0) GO TO 10
```

```
C
C PUT THE ID INTO THE CONNECT LIST
C
15     NUMCON=NUMCON+1
       IF (OUTFLG.EQ.1.) IDTABL(IDFLAG,MAXID2)=1
       IF (OUTFLG.EQ.0.) IDTABL(IDFLAG,MAXID1)=1
       IF (NUMCON.GT.MAXCON) GO TO 700
       DO 20 I=1,MAXID
          LSTCON(NUMCON,I)=LINE1(I)
20     CONTINUE
C
C CHECK FOR DUPLICATE IDS IN THE CONNECT LIST
C
       CALL CONCHK
       GO TO 10
C
C DETERMINE WHERE THE OUTLIST ENDS AND THE
C INLIST BEGINS...PERFORM AN INFO DUMP
C
400    IF (NUMOUT.EQ.0) NUMIN=NUMCON
       IF (NUMOUT.NE.0) NUMIN=NUMCON-NUMOUT
       WRITE (2,410) NUMCON,NUMOUT,NUMIN
       WRITE (5,410) NUMCON,NUMOUT,NUMIN
410    FORMAT (/,' CONECT 410--',/,
     1           ' TOTAL # NODES COMMON TO SALOGS=',I10,
     1        /,' OUTPUT NODES TO SALOGS=          ',I10,
     2        /,' INPUT NODES FROM SALOGS=         ',I10)
       IF (NUMCON.EQ.0) GO TO 475
       IF (NUMOUT.GT.0) WRITE (2,415)
     1 ((LSTCON(I,J),J=1,MAXID),I=1,NUMOUT)
       IF (NUMOUT.GT.0) WRITE (5,415)
     1 ((LSTCON(I,J),J=1,MAXID),I=1,NUMOUT)
415    FORMAT (//,' *** OUTLIST ***',/,100(T2,8A1,/))
       IF (NUMIN.GT.0) K=NUMOUT+1
       IF (NUMIN.GT.0) WRITE (2,420)
     1 ((LSTCON(I,J),J=1,MAXID),I=K,NUMCON)
       IF (NUMIN.GT.0) WRITE (5,420)
     1 ((LSTCON(I,J),J=1,MAXID),I=K,NUMCON)
420    FORMAT (//,' *** INLIST ***',/,100(T2,8A1,/))
475    RETURN
```

```
C
C ERROR MESSAGES
C
500   WRITE (2,501)
      WRITE (5,501)
501   FORMAT (' CONECT 501-- THE 1ST CARD MUST BE A ',
     1 'CONECT CARD')
      CALL HALT
600   WRITE (2,601)
      WRITE (5,601)
601   FORMAT (' CONECT 601-- A BLANK MUST FOLLOW ',
     1 '"CONECT"')
      CALL HALT
700   WRITE (2,701)
      WRITE (5,701)
701   FORMAT (' CONECT 701-- TOO MANY NODES IN COMMON WITH',
     1 ' THE SALOGS MODEL')
      CALL HALT
800   WRITE (2,801)
      WRITE (5,801)
801   FORMAT (' CONECT 801-- A BLANK MUST FOLLOW ;')
      CALL HALT
      END
```



```
C
C
C
      SUBROUTINE [illegible]
C
      COMMON [illegible] ,IFLAGS,I[illegible],
     1 LIN[illegible]
      COMMON [illegible] ,NTOT [illegible],NUMCON,
     1 ICOUNT
C
C PURPOSE [illegible] DUPLICATE ENTRIES IN THE [illegible]
C CONSTRAINT [illegible].
C
      INTEG[illegible]
      IF (I[illegible]) GO TO 1000
      DO 5 I 2,[illegible]
         DO 3 J [illegible]
            IF ( [illegible] (I,IN.[illegible]ON(ICON,J)) GO TO 5
3        CONTINUE
         WRITE (2,4) [illegible](I,[illegible])
         WRITE (3,[illegible](I,[illegible])
4        FORMAT (' [illegible] ENTRY ',8A1,
     1              ' ON THE CONTROL CARD')
         NUMCON [illegible]
         GO TO 1000
5     CONTINUE
1000  RETURN
      END
```

```
[illegible]

SUBROUTINE [illegible]

C [illegible]
C [illegible]. THIS [illegible]

C [illegible]

C COMPUTE [illegible] LETTER

C [illegible]
C

[illegible]

[illegible]
```

```
C
C
C
      FUNCTION IBKPUT(NUMHSH)
C
      COMMON/SYSB/[illegible],LINE1(80),IBLANK,IALPHA,[illegible],
     1 L[illegible],ID[illegible],[illegible]DEIL
      COMMON/NAME/[illegible](69,12),[illegible],MAXNM1,MAXNM2,MAXNM3,
     1 MAXNM4,MAXBL[illegible]CHR(70),[illegible]CHR
C
C WILL DEVELOP A HASH NUMBER BASED ON THE FIRST NUMHSH
C CHARACTERS IN LINE1. THIS ROUTINE IS USED FOR BLOCK NAMES.
C
      IBKPT1=0
      DO 10 I=1,NUMHSH
         IF (LINE1(I).EQ.IBLANK) GO TO 100
         ICHR=LINE1(I)
         IBKPT1=ICHAR(ICHR)+10+I*MAXNAM+IBKPT1
10    CONTINUE
100   IBKPT1=MODULO(IBKPT1,MAXBLK)
      IBKPUT=IBKPT1
      RETURN
      END
```

```
C
C
C
      FUNCTION IDHASH(IDUMMY)
C
      COMMON [illegible] ),IBLA[illegible],EXTRA,[illegible],
     1 [illegible],MAXID,[illegible]
      COMMON /IDS/ IDTABL(1000,10),IDPLAC,MAXID1,MAXID2,
     1 IDTSIZ
C
C HASHES THE ID FOUND IN LINE1 INTO THE IDTABL ARRAY
C
      IPLACE=IDPUT(MAXID)
      ICYCLE=0
      LIMIT=IDTSIZ
5     ICYCLE=ICYCLE+1
      DO 10 I=IPLACE,LIMIT
         IF (IDTABL(I,MAXID1).EQ.IBLANK) GO TO 50
         DO 7 J=1,MAXID
            IF (LINE1(J).NE.IDTABL(I,J)) GO TO 10
7        CONTINUE
         GO TO 60
10    CONTINUE
      IF (ICYCLE.EQ.2.OR.
     1   (ICYCLE.EQ.1.AND.IPLACE.EQ.1)) GO TO 500
      LIMIT=IPLACE-1
      IPLACE=1
      GO TO 5
50    IDTABL(I,MAXID1)=0
      IDTABL(I,MAXID2)=0
      DO 55 J=1,MAXID
         IDTABL(I,J)=LINE1(J)
55    CONTINUE
60    IDHASH=I
      RETURN
500   WRITE (2,501)
      WRITE (5,501)
501   FORMAT (' IDHASH 501-- CANT HASH ANYMORE IDS')
      CALL HALT
      END
```

```
                         <ENTRY POINT>
                               V
                               V
                               V
                               V
 ----------------------------------------------------------------
 | C                                                            |
 | C                                                            |
 | C                                                            |
 |       [illegible]                                            |
 | C                                                            |
 | C [illegible] THE FIRST [illegible]                          |
 | C [illegible]. THIS ROUTINE IS USED ONLY FOR NODES.          |
 | C                                                            |
 | C                                                            |
 | C RESET HASH COUNTER                                         |
 | C                                                            |
 | C COMPUTE THE HASH COUNT DEPENDING ON LETTER AND POSITION.   |
 | C                                                            |
 | C COMPUTE THE MODULO OF THE HASH COUNT.                      |
 | C                                                            |
 ----------------------------------------------------------------
                               V
                               V
                               V
                               V
 ----------------------------------------------------------------
 |       RETURN                                                 |
 ----------------------------------------------------------------

 ----------------------------------------------------------------
 |       END                                                    |
 ----------------------------------------------------------------
```

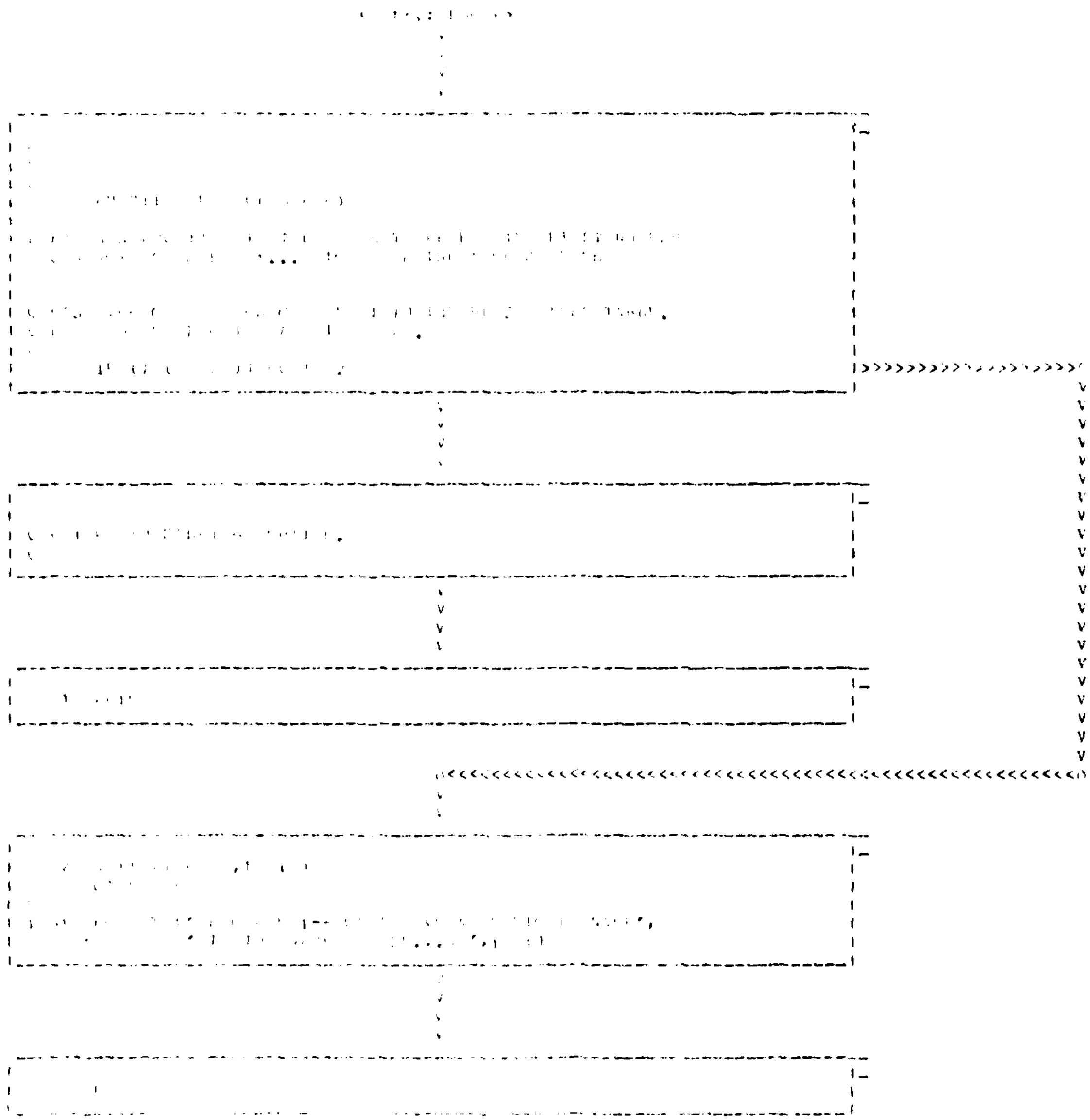
```
C
C
C
      FUNCTION IDPUT(NUMHSH)
C
      COMMON /[illegible]/ IDTABL([illegible],10),IDP[illegible],MAXID1,MAXID2,
     1 IDTSIZ
      COMMON/[illegible]/ LINE(80),LINE1(80),IBLANK,IASTRA,MAXLIN,
     1 LINE[illegible],IO[illegible]R,MAXID,[illegible],ERR[illegible]
C
C WILL DEVELOP A HASH NUMBER BASED ON THE FIRST NUMHSH
C CHARACTERS IN LINE1. THIS ROUTINE IS USED ONLY FOR NODES.
C
      IDPUT1=0
      DO 10 I=1,NUMHSH
         IF (LINE1(I).EQ.IBLANK) GO TO 100
         ICHR=LINE1(I)
         IDPUT1=ICHAR(ICHR)*10+I*MAXID+IDPUT1
10    CONTINUE
100   IDPUT1=MODULO(IDPUT1,IDTSIZ)
      IDPUT=IDPUT1
      RETURN
      END
```

```
C
C
C
      FUNCTION MODULO(NUMBER,MODIN)
C
C     [illegible]
C
C [illegible] REMAINDER OF NUMBER/MODIN
C [illegible] ALL ARGUMENTS ARE INTEGER
C
      FMODIN=ABS(FLOAT(MODIN))
      IF (FMODIN.LT.1.) GO TO 500
      VALUE=ABS(FLOAT(NUMBER))/FMODIN
      IVALUE=INT(VALUE)
      MODULO=INT((VALUE-FLOAT(IVALUE))*FMODIN)
      RETURN
500   WRITE (2,501) MODIN
      WRITE (5,501) MODIN
501   FORMAT (' MODULO 501--- CAN ONLY USE A MODULUS>0.',
     1        ' PRESENT MODIN=',I10)
      CALL HALT
      END
```

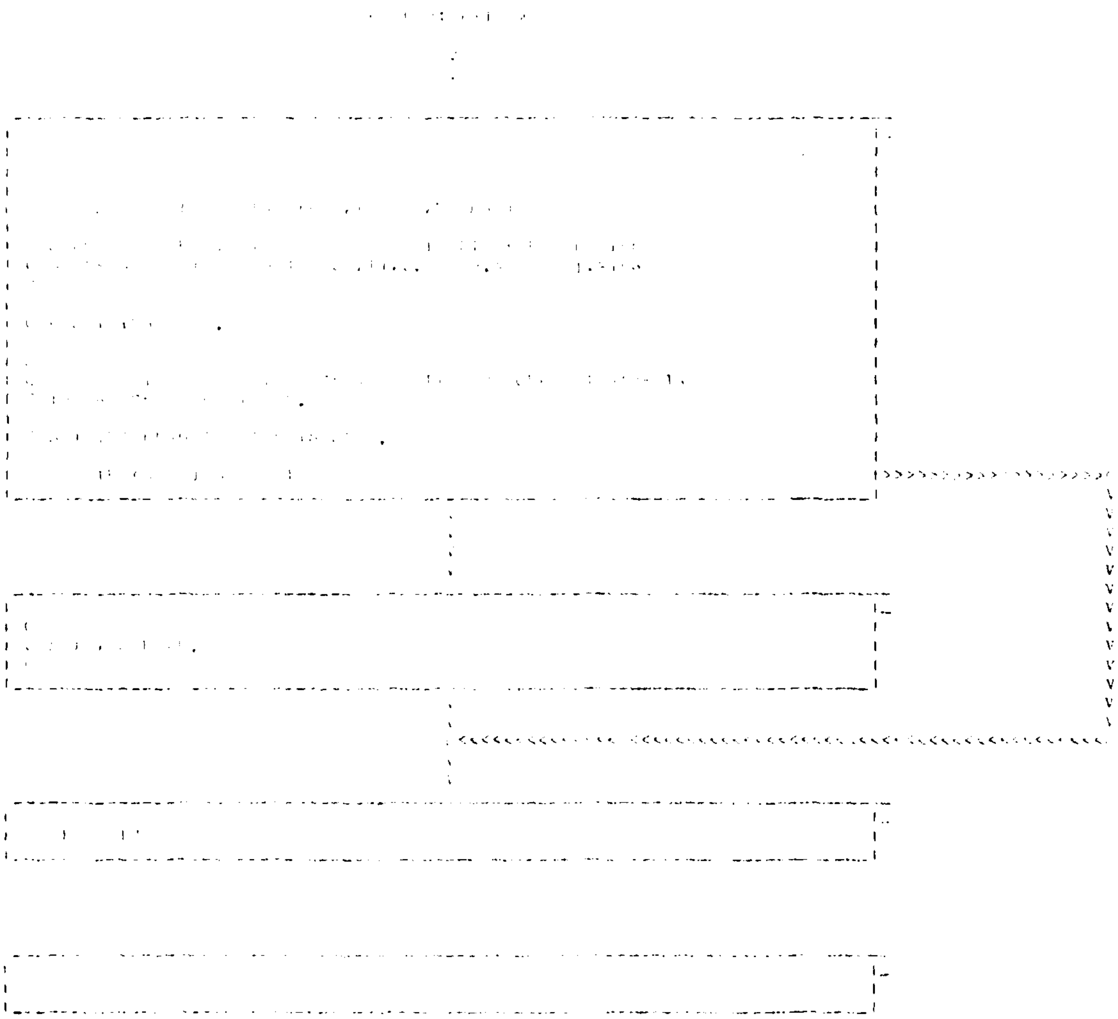
```
&lt;[illegible] IC [illegible]&gt;
   V
   V
   V
   V
------------------------------------------------------------
| C
| C
| C
|      [illegible] IC[illegible] (IC[illegible])
| C
|   [illegible] CHARACTER.
| C
| C
| C [illegible] AND [illegible] ITS POSITION
| C IN THE CHARACTER [illegible] LIST.
| C
------------------------------------------------------------
   V
   V
   V
   V
------------------------------------------------------------
|      RETURN
------------------------------------------------------------

------------------------------------------------------------
|      END
------------------------------------------------------------
```

```
C
C
C
      FUNCTION [illegible](CHR)
C
      [illegible]
     1 [illegible]
C
C CON[illegible] THE [illegible] TO [illegible]
C
      DO 10 I=1,[illegible]
         IF ([illegible](I).EQ.ICH[illegible]) GO TO 20
10    CONTINUE
      I=[illegible]
20    ICHAR=I
30    RETURN
      END
```

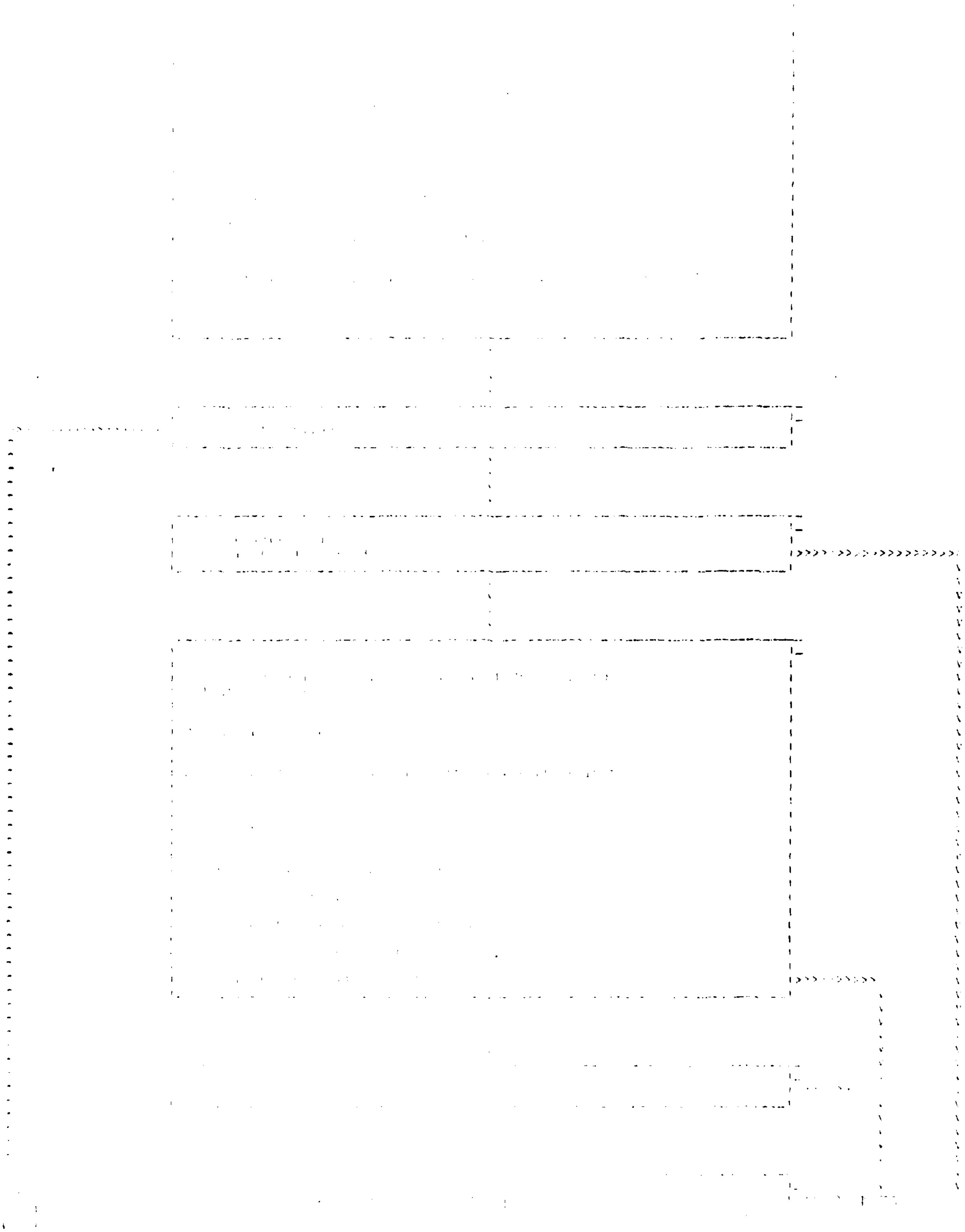
```
C
C
C
      [illegible]
C
      [illegible]
     1 [illegible]
      [illegible]
     1 [illegible]
C
C FINDS [illegible] THE BLOCK NAME LOCATED IN LIST1 IS IN THE
C NAMES [illegible]. IFIND=0 IF IT IS NOT FOUND.
C
      IFIND=0
      ICYCLE=0
      LIMIT=MAXBLK
      IPLACE=[illegible](CU[illegible]H)
5     ICYCLE=ICYCLE+1
      DO 10 I=IPLACE,LIMIT
         IF (NAMES(I,MAXNM2).EQ.IBLANK) GO TO 200
         II=I
         CALL EXIST(ANS,II,NUMBER)
         IF (ANS.EQ.1) GO TO 100
10    CONTINUE
      IF (ICYCLE.EQ.2.OR.
     1   (ICYCLE.EQ.1.AND.IPLACE.EQ.1)) GO TO 200
      LIMIT=IPLACE-1
      IPLACE=1
      GO TO 5
100   IFIND=1
200   RETURN
      END
```

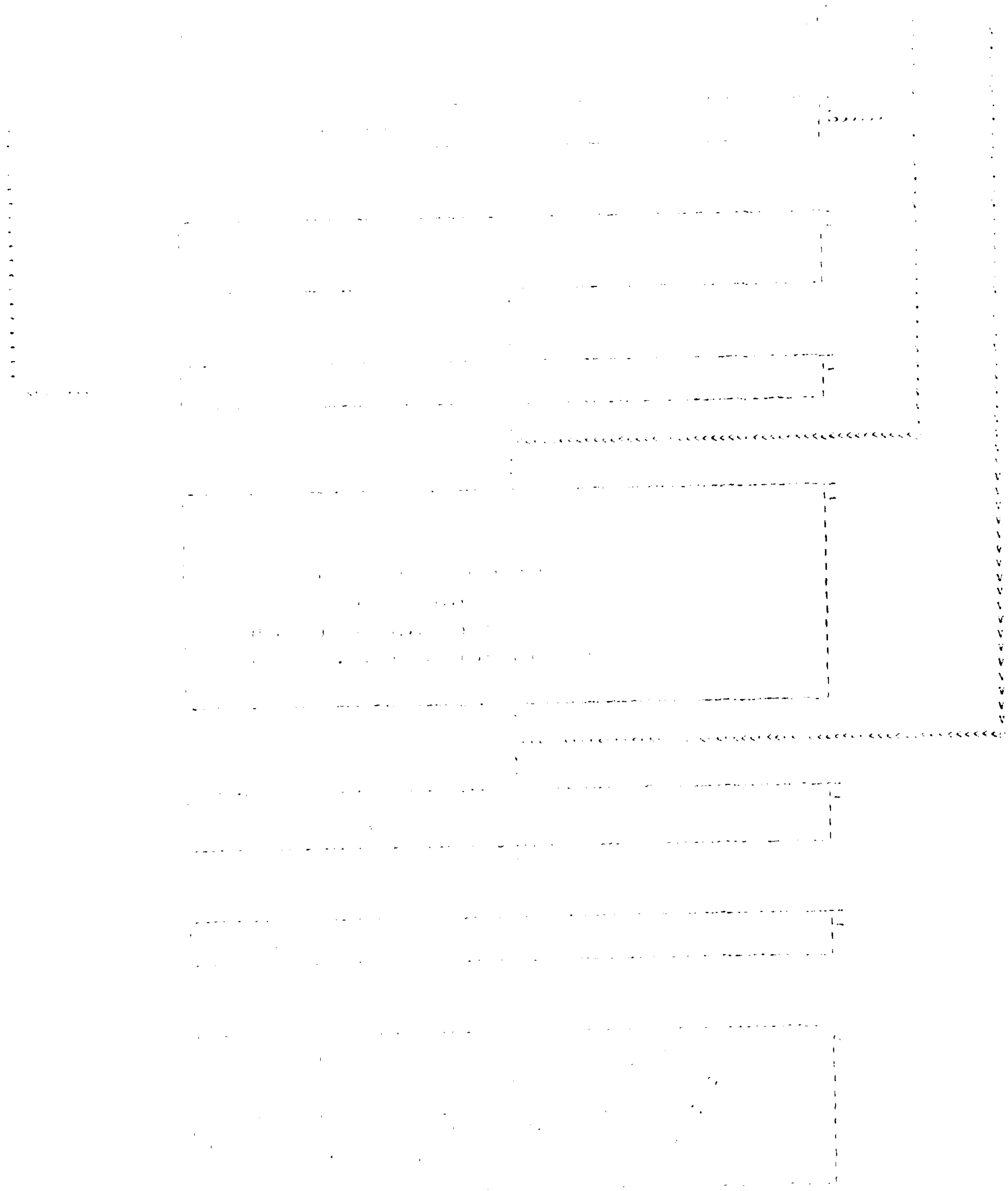


```
C
C
C
      FUNCTION [illegible]
C
      [illegible]
     1 [illegible]
      [illegible]
     1 [illegible]
C
C FINDS A [illegible] IN [illegible] FOR THE [illegible]
C CHARACTERS OF [illegible]1...WHICH [illegible] A BLOCK [illegible]
C
      LOCAL=0
      ICYCLE=0
      LIMIT=[illegible]
      IPLACE=[illegible](NUMSH)
5     ICYCLE=ICYCLE+1
      DO 10 I=IPLACE,LIMIT
         IF (NAM[illegible](I,[illegible]).EQ.IBLANK) GO TO 100
         I1=I
         CALL [illegible](ANS,I1,NUMSH)
         IF (ANS.EQ.1.) GO TO 500
10    CONTINUE
      IF (ICYCLE.EQ.2.OR.
     1   (ICYCLE.EQ.1.AND.IPLACE.EQ.1)) GO TO 200
      LIMIT=IPLACE-1
      IPLACE=1
      GO TO 5
100   LOCAL=I
200   RETURN
500   WRITE (2,501) (LINE1(I),I=1,NUMSH)
      WRITE (5,501) (LINE1(I),I=1,NUMSH)
501   FORMAT (' LOCAL 501-- CANT HAVE A DUPLICATE NAME',
     1         ' IN THE NAME TABLE...= ',10A1)
      CALL HALT
      END
```



```
C
C
C
      SUBROUTINE [illegible] (A[illegible],IPLACE,[illegible])
C
      C[illegible]
     1[illegible]
      C[illegible],KA[illegible],KANNEB,
     1[illegible],N[illegible]
C
C C[illegible] TO S[illegible] IF T[illegible] BLOCK [illegible] IN LI[illegible] EQUALS THE
C B[illegible] GATE AT [illegible](IPLACE,I)...     0.=NO     1.=YES
C
      ANS=0.
      DO 10 I=1,N[illegible]
         IF (JA[illegible](IPLACE,I).NE.LINE1(I)) GO TO 100
10    CONTINUE
      ANS=1.
100   RETURN
      END
```





```
C
C
C
      SUBROUTINE READEM
C
      [illegible]),[illegible],[illegible],MAXNM4,
     1 [illegible],[illegible],[illegible]
      COMMON/[illegible]/[illegible],[illegible]([illegible]),[illegible],LASTNA,MAXLIN,
     1 LINE[illegible],[illegible],[illegible],[illegible]
C
C READS ALL THE BEHAVIORAL BLOCK NAMES AND THE NUMBER OF
C OUTPUT/INPUT NODES REQUIRED FOR EACH. THESE ARE HASHED
C INTO THE NAMES TABLE. THE INPUT DATA FOR THIS
C ROUTINE SHOULD RESIDE IN SIMU.NAM AND IS ENTERED IN A
C FORMATTED FORM--> COL 1-8 NAME-BLOCK NAME (LEFT JUSTIFIED)
C                   COL 1-5 NUMBER OUTPUT NODES REQUIRED
C                   COL 6-10 NUMBER INPUT NODES REQUIRED
C                   COL 11-15 FUNCTIONAL SUBROUTINE NUMBER
C                             THIS IS THE ? OF FNUM?
C (THIS IS A TWO LINE FORMAT:
C
C   LINE 1 IS LEFT JUSTIFIED
C   LINE 2 NUMBERS ARE RIGHT JUSTIFIED
C
      IF (MAXBLK.GT.MAXLIN) GO TO 700
      WRITE (2,5)
      WRITE (5,5)
5     FORMAT (/,' READEM 5--',/,' BLOCK NAME',5X,
     1        '# OUTPUTS',5X,'# INPUTS',5X,'# LINES',5X,
     2        'FNUM?',5X,'HASH #')
C
C LOOP AROUND TO GET ALL BEHAVIORAL BLOCK NAMES AND
C THEIR PARAMETERS
C
      DO 50 N=1,MAXBLK
         READ (20,10,END=100) (LINE1(I),I=1,MAXNAM),NOUT,NINPUT,
     1                        NFNUM
10       FORMAT (8A1,/,3I5)
C
C HASH BLOCK NAME
C
         IPLACE=LOCAL(MAXNAM)
         IF (IPLACE.EQ.0) GO TO 53
         DO 15 I=1,MAXNAM
            NAMES(IPLACE,I)=LINE1(I)
15       CONTINUE
C
C PLACE BLOCK INFORMATION INTO ITS PROPER PLACE IN THE
C NAMES TABLE
C
         NAMES(IPLACE,MAXNM1)=NOUT
         NAMES(IPLACE,MAXNM2)=NINPUT
         NAMES(IPLACE,MAXNM3)=NOUT+NINPUT
         NAMES(IPLACE,MAXNM4)=NFNUM
         WRITE (2,40) (NAMES(IPLACE,J),J=1,MAXNM4),IPLACE
         WRITE (5,40) (NAMES(IPLACE,J),J=1,MAXNM4),IPLACE
40       FORMAT (T2,8A1,11X,I5,8X,I5,7X,I5,5X,I5,6X,I5)
```

```
C
C [illegible]
C
          IF ([illegible](IPLACE,[illegible]).LT.1.[illegible]
     1    [illegible].1.
     2    [illegible].1[illegible]
     3    [illegible](IPLACE,[illegible]).GT.[illegible]) GO TO 450
          IF ([illegible](I[illegible],[illegible]4)).[illegible]0) GO TO 500
          LINE([illegible](IPLACE,[illegible])) 1
          DO 45 J=1,[illegible]
             IF ([illegible](IPLACE,J).EQ.IBLANK) GO TO 50
             ICHR=[illegible](IPLACE,J)
             IF (IC[illegible]R(ICHR).GT.36) GO TO 600
45        CONTINUE
50     CONTINUE
C
C ERROR MESSAGES
C
53     WRITE (2,55) MAXBLK
       WRITE (5,55) MAXBLK
55     FORMAT (' READUM 55-- TOO MANY BEHAVIORAL UNITS',
     1          ' MAXBLK=',I10)
       CALL HALT
100    WRITE (2,210)
       WRITE (5,210)
210    FORMAT (1X)
       RETURN
450    WRITE (2,451) MAXBLK
       WRITE (5,451) MAXBLK
451    FORMAT (' READUM 451-- ONE OF THE ABOVE PARAMETERS',
     1          ' =0 OR THE ? IN FNUM?>',I5)
       CALL HALT
500    WRITE (2,501)
       WRITE (5,501)
501    FORMAT (' READUM 501-- THE ? IN THE ABOVE FNUM? IS A',
     1      ' DUPLICATE')
       CALL HALT
600    WRITE (2,601)
       WRITE (5,601)
601    FORMAT (' READUM 601-- THE ABOVE BLOCK NAME HAS',
     1          ' INVALID CHARACTERS (A-Z,0-9 ONLY)')
       CALL HALT
700    WRITE (2,701)
       WRITE (5,701)
701    FORMAT (' READUM 701-- SYSTEM ERROR, MAXBLK EXCEEDS',
     1          ' MAXLIN. CANT USE LINE ARRAY FOR FNUM?',
     2          ' DUPLICATION CHECK')
       CALL HALT
       END
```

APPENDIX D

SALOCS USERS GUIDE

FUNDED BY

UNITED STATES DEPARTMENT OF ENERGY

NOTICE

THIS REPORT WAS PREPARED AS AN ACCOUNT OF WORK SPONSORED BY THE UNITED STATES GOVERNMENT. NEITHER THE UNITED STATES NOR THE UNITED STATES DEPARTMENT OF ENERGY, NOR ANY OF THEIR EMPLOYEES, NOR ANY OF THEIR CONTRACTORS, SUBCONTRACTORS, OR THEIR EMPLOYEES, MAKES ANY WARRANTY, EXPRESS OR IMPLIED, OR ASSUMES ANY LEGAL LIABILITY OR RESPONSIBILITY FOR THE ACCURACY, COMPLETENESS OR USEFULNESS OF ANY INFORMATION, APPARATUS, PRODUCT OR PROCESS DISCLOSED, OR REPRESENTS THAT ITS USE WOULD NOT INFRINGE ON PRIVATELY OWNED RIGHTS.

S A L O G S   U S E R ' S   G U I D E

JERRY D. [illegible]

```
         A
   A    A A    A
    AA AA AA AA
     AAA   AAA
  AAAAAAAAAAAAAAAA
      AAA AAA
      AA   AA
     A       A
```

SALOGS IV IS THE SANDIA LOGIC CIRCUIT SIMULATOR. THE LOGIC CIRCUIT TO BE SIMULATED MAY CONTAIN LOGIC GATES, LIBRARY LOGIC MODELS, USER DEFINED LOGIC MODELS, AND/OR USER DEFINED FUNCTIONAL MODELS. SIMULATION IS CONTROLLED BY SPECIFYING INPUTS TO THE CIRCUIT, TIME STEPS, CONDITIONS OF SIMULATION, AND WHAT IS TO BE PRINTED OUT DURING OR AFTER SIMULATION.

THE LOGIC MODELS ARE WRITTEN IN A NETWORK DESCRIPTION LANGUAGE(NDL), THE FUNCTIONAL MODELS CAN BE WRITTEN IN FORTRAN, AND THE SIMULATION CONTROL IS WRITTEN IN SALSIM.

SALOGS HAS BUILT-IN LOGIC GATE DEFINITIONS FOR THE OPERATIONS OF: AND, OR, NAND, NOR, EXCLUSIVE-OR, INVERSION, WIRED-OR, WIRED-OR WITH A PRIORITY, TRANSMISSION GATES, BUFFERS, AND MULTIPLEXERS. LOGIC GATES, LOGIC MODELS, AND FUNCTIONAL MODELS MAY BE FREELY INTERMIXED IN A GIVEN CIRCUIT.

[illegible]

B6. [illegible]

C) SALORS [illegible]

C1. [illegible]
C2. [illegible]
C3. [illegible]
C4. EXAMPLE [illegible]
C5. [illegible]
C6. SALSYM [illegible] CODES.

A) LOG[illegible] STAT[illegible]

[illegible]

0 – I[illegible]
A – C[illegible]
H – T[illegible]
I – H[illegible]

....

D – D[illegible] OR [illegible] SH[illegible].
S – T[illegible]
T[illegible] – TR[illegible]
U – UP[illegible] POSITI[illegible].

T[illegible]
I[illegible], [illegible] PLOTTE[illegible]. TH[illegible]
$DO[illegible], [illegible] THE[illegible]
AR[illegible] (E.G. TH[illegible] $IS[illegible]
FI[illegible] IS C[illegible] BY [illegible]
IN[illegible] AS A T[illegible] RE[illegible].)
USER CAN COP[illegible] IT TO HIS [illegible] IT OR USE IT AS A T[illegible]

[illegible]

B1. [illegible]

[illegible] ALL OF [illegible] [illegible] RECORD.

EXA[illegible]S:
INP[illegible] X1 X[illegible] X3
INP[illegible] A,B,C
OUTPU[illegible] Y,Z
OU[illegible] ,Y,Z
OUTP[illegible] W Y Z

B2. BASIC GAT[illegible]

THE BASIC [illegible] AND, [illegible] XOR, [illegible] ELEMENT [illegible], XOR, [illegible] WITH [illegible] AND NOR [illegible] GATES [illegible] HAV[illegible] ONLY ONE [illegible]. THE [illegible] 4 [illegible] AND THE REST OF [illegible] THE FORM: GATE NAME OUTPUT NO[illegible] INPUT [illegible]

EXAMPLES:
AND OU[illegible] X1 X2 X3
AND Y1 X1 X2
OR [illegible] X1 X2 X3 [illegible] X5
N[illegible] A,B,C,D,[illegible]
T[illegible] DATA [illegible] INPUT
XOR O[illegible] A B
I[illegible] A
BUF [illegible] C

[illegible]

FOR [illegible] [illegible] -1 -2 -1 -2. [illegible]

A DELAY WITH A 2 [illegible] -2.

B4. LOGIC MODELS.

[illegible]

1	[illegible]
2	[illegible]
3	[illegible]
4	[illegible]
5	[illegible]
6	[illegible] (... FOR LOGIC MODELS)
7	[illegible]

[illegible]

[illegible]

```
C1[illegible]            1     1     [illegible]     3     0     1
[illegible]   5   A   B
      2   3   [illegible]
   [illegible]   2   3   4
END C[illegible]
C[illegible])            2     1     [illegible]     3     0     1
[illegible]   X   [illegible]   [illegible]
      2   3   4
   [illegible]   5   5   6
   DEL   2   5   -1   -2
END C[illegible]
C(14,2)          14     1     3     4     0     1
    2     4     3     5
   INV 13   3
   [illegible]   7 13   4
   [illegible]   8 25   7
   WORP   9   8   6
   INV 14   9
   INV 10 14
   DEL   6 10   -1   -2
   TGB 17   3 14
   WORP 19 17 16
   INV   2 19
   INV 20   2
   DEL 16 20   -1   -2
   INV 15   5
   TGB 25 13 15
END C14,2)
$END MODELS
```

B5. FUNCTIONAL MODELS.

FUNCTI[illegible] TO [illegible] COMPLEX CIRCUITS [illegible]. [illegible] MODELS ARE WRITTEN AS FORTRAN SUBROUTI[illegible], WHERE ? IS AN INTEGER VALUE FROM 1 THROUGH 60. THE FO[illegible] IS [illegible] AT [illegible]. THE FUNCTIONAL MODEL [illegible] IN THE USER MODELS PART OF [illegible] ($[illegible]S AND $END MODELS). THE FORMAT FOR [illegible] IS THE SAME AS FOR LOGIC MODELS, [illegible] EXCEPT THAT THE FUNCTIONAL MODEL [illegible] THE NUMBER USED IN THE SUBROUTINE [illegible] ABOVE.) THE FUNCTIONAL MODEL USES [illegible] RECORDS TO IDENTIFY IT; THE HEADER, I/O LIST, AND END RECORDS.

[illegible]

THE [illegible] TO [illegible]
[illegible]
[illegible] SLOPE.
[illegible]

0 [illegible] 0 - [illegible]
1 FOR [illegible] - [illegible] ([illegible])
2 FOR H - [illegible].
3 FOR [illegible] - [illegible].
4 FOR [illegible] - [illegible] SLOPE.
5 FOR X - TRA[illegible].
6 FOR A - [illegible].
7 FOR [illegible] - [illegible].

THE [illegible] TO GRAY CODE
CONV[illegible].

```
      SUBRO[illegible]
C
C         TEST OF [illegible].
C
      COMM[illegible](40)
C
C     OUTPUT([illegible] CODE)      = INP[illegible].
C
      JO[illegible]
      JO[illegible]
      JO[illegible](3)[illegible](3)[illegible]
      RETURN
      END
```

B6. EXAMPLE CIRCUIT.

```
[illegible]S
C[illegible]20          1    1    2    3    0    1
*     X   A   B
      2   3   4
   [illegible]   2   3   4
[illegible]
C[illegible]20          2    1    2    3    0    1
*     X   A   B
      2   3   4
   [illegible]N   3   3   4
   [illegible]   2   [illegible]   1   [illegible]
[illegible]
```

```
[illegible]
[illegible]  5  1  6  7  0  1
  4  3  2  5  6  7  8
    [illegible]  3  0  2  3
    [illegible]  10  6  [illegible]
    [illegible]  11  8  7
    [illegible]  12  9  10
    [illegible]  4  1[illegible]  11
END [illegible]
CD[illegible])  4  2  2  [illegible]  0  1
*  Q  QR  SR  R[illegible]
      4  3  5  2
    N[illegible]  7  5  3
    R[illegible]  6  2  4
    [illegible]  4  7  -1  -2
    D[illegible]  3  6  -3  -4
END (C[illegible])
C[illegible])  4  2  2  4  0  1
*  Q  QR  S  R
    5  3  4  2
    N[illegible]  5  2  6
    D[illegible]  6  3  1  1
    R[illegible]  3  4  7
    D[illegible]  7  5  1  1
END [illegible]
C[illegible]  4  2  2  4  0  1
*  Q  QR  S  R
    [illegible]  2  4  2
    N[illegible]  3  2  3
    D  3  0  -3  -5
    R[illegible]
    D[illegible]  5  8  -1  -2
END [illegible]
C[illegible]  1[illegible]  1  3  5  0  1
  [illegible]  4  3  5  6
    [illegible]  13  3
    [illegible]
    [illegible]  7
    [illegible]  9  8  6
    [illegible]  11  9
    [illegible]  10  14
    D[illegible]  6  10  1
    [illegible]  17  [illegible]
    [illegible]  1[illegible]  1
```

```
    INV  2 19
    HET 20  2
    0.5 1 20 -1   2
    [illegible]
    [illegible] 15 15
[illegible]
[illegible]
IN[illegible]
OUT[illegible]
C12[illegible]
C12[illegible]
C1230 [illegible] X3 [illegible]
C12[illegible] X2 X4 [illegible]
C1230 [illegible]
INV  [illegible] X1
END
```

C) [illegible] LANGUAGE ([illegible]).

[illegible]

[illegible] 72 [illegible] ACCORDING TO [illegible]:

POSITION	FIELD
1-8	LABEL
9	CONTINUATION MARK
10-72	INSTRUCTION

A STATEMENT IS CONSIDERED A CONTINUATION OF THE PREVIOUS RECORD IF ANY [illegible] CHARACTER IS IN POSITION 9. AN ASTERISK (*) AS [illegible] INDICATES A COMMENT. STATEMENT LABELS [illegible] 1 TO 6 ALPHABETIC CHARACTERS. IN GENERAL, COMMAS ARE DELIMITERS FOR NODE NAMES. LOGICAL CONDITIONS ARE ENCLOSED IN PARENTHESIS AND USE THE FORTRAN-LIKE OPERATORS OF EQ, NE, OR, GT, LT, AND, EOR, LE, NOT, AND GE, SET OFF BY PERIODS. LOGICAL OPERATIONS CAN BE COMPOUNDED, FOR EXAMPLE:

```
IF ((A1.EQ.$$STATE).OR.(TIME.GE.63)) STOP
```

C1. INPUT CONTROL COMMANDS

THE INPUT CONTROL COMMANDS SET THE NODES OF THE CIRCUIT TO ONE OF THE EIGHT LOGICAL STATES. THE COMMANDS IN THIS GROUP ARE:

IT? -WHERE ? CAN BE 0,*,P,1,D,X,A, OR U. THIS COMMAND INITIALIZES ALL OF THE NODES IN THE CIRCUIT TO THE SPECIFIED STATE. BY DEFAULT, THE CIRCUIT IS ENTIRELY INITIALIZED TO UNDEFINED(*) AT THE BEGINNING OF EACH SIMULATION.

RESTART -THIS COMMAND IS USED TO RESTART THE SIMULATION WITH ALL OF THE CIRCUIT NODES INITIALIZED TO THE STATES DETERMINED BY THE LAST DUMP COMMAND(IN THE CURRENT OR A PRIOR SIMULATION RUN.) THIS ACTUALLY READS IN THE NODE STATES FROM A FILE CREATED BY THE DUMP COMMAND.

RESTORE -THE RESTORE COMMAND RETURNS THE SPECIFIED NODE(S) TO CIRCUIT CONTROL. CONSIDER A NAND GATE WITH INPUTS X1 AND X2 AND AN OUTPUT F WITH F FROZEN AT 0 BY A SET TO COMMAND. THE STATE OF F WILL REMAIN AT 0 INDEPENDENT OF THE VALUES ON THE NODES X1 AND X2. AFTER A RESTORE F COMMAND THE NAND GATE AND THE INPUTS X1 AND X2 WILL DETERMINE THE STATE OF NODE F.

SEQUENCE -THIS COMMAND APPLIES ONLY TO CIRCUIT INPUT NODES. THE SEQUENCE COMMAND ESTABLISHES A BINARY INPUT SEQUENCE FOR THE SPECIFIED CIRCUIT INPUT NODES. THE FORMAT IS:
SEQ [INITIAL VALUE] (B1,B2,...,BN) NODE NAMES.

[illegible]

[illegible]

[illegible]	4 STA[illegible]	[illegible]
SEQ 1 (1,[illegible]	100[illegible]	100[illegible]
SEQ [illegible] ([illegible] X3	0011[illegible]	00[illegible]
SEQ 1 ([illegible] X[illegible]	1[illegible]	11[illegible]
X[illegible]	1100[illegible]	11[illegible]

THE S[illegible] CO[illegible]AND [illegible] ALSO [illegible] TO SET UP A B[illegible]ARY
COUNT [illegible]:
SEQ BI[illegible] ([illegible],[illegible],...,[illegible]) [illegible],X2,X3 [illegible] COU[illegible], [illegible],
ST[illegible] [illegible] SIGNIFIC[illegible] BIT.
BIND [illegible].
EXAMPLES:

SEQ BI[illegible] (1,2,3) X1,X2

TIME	X1	X2
1	0	0
2	0	1
3	1	0
4	1	1
5	0	0
6	0	1

SEQ BIND (1,2,3) A,B

TIME	A	B
1	1	1
2	1	0
3	0	1
4	0	0
5	0	0
6	0	0

SEQ BIND (1,2) A,B

TIME	A	B
1	1	1
2	1	0
3	0	1
4	0	0
5	0	0
6	0	0

SET TO -THE SET TO CO[illegible]AND [illegible] THE [illegible] NODE(S) TO THE GIV[illegible] [illegible](S.) THIS NODE WILL RETAIN [illegible] THE SPECI[illegible] [illegible] UNTIL A RESTORE OR [illegible] CO[illegible]AND IS ENCOU[illegible] FOR [illegible] NODE. IF THERE AR[illegible] STATES [illegible] THAN NODES LISTED THE EXTRA STATES ARE IG[illegible]. IF [illegible] MORE [illegible] THAN STATES SPECIFIED THE LAST STATE WILL BE USED FOR ALL OF [illegible] EXTRA NODES.

EXAMPLES:

SET TO 1 A,B,C	A,B, AND C WILL BE SET TO 1.
SET TO [illegible] OUT	OUT WILL BE MADE UNDEFINED.
SET TO [illegible] [illegible],X1	BUS IS [illegible]T TO HIGH IMPEDANCE, X1 IS SET TO 1, AND 0 IS IGNORED.

C2. OUTPUT CONTROL C[illegible]DS

[illegible] OUTPUT (I[illegible] OR [illegible]) [illegible] CREATED [illegible]

[illegible] CONTROL. [illegible]

HAZARD (ON OR OFF) - TURNS ON OR OFF THE [illegible] ANALYSIS. THE HAZARD [illegible] CONT[illegible] [illegible] NODES IN THE TRANSIT[illegible] [illegible].

HP -'HALT PRINTING' STOPS THE PRINTING OF THE NODE STATES SPECIFIED IN THE LAST PRINT STATEMENT.

PC -'PRINT COMMENT' PRINTS THE MESSAGE ([illegible] OF 80 CHARACTERS) FOLLOWING PC IN THE SIMULATION OUTPUT.

PCO -'PRINT CHANGES ONLY' PRINTS THE NODE STATES SPECIFIED IN THE LAST PRINT STATEMENT ONLY WHEN SOME NODE IN THE CIRCUIT CHANGES STATE.

PE (M+NT) -PRINTS THE NODE STATES EVERY N TIMESTEPS AFTER M TIMESTEPS.

PRINT NODE NAMES -PRINTS THE STATES OF SPECIFIED NODES.

PV? -PRINTS THE LIST OF NODES IN THE STATE ?, E.G. PV* WILL LIST ALL THE NODES IN THE UNDEFINED STATE.

SP -STARTS PRINTING THAT HAS BEEN SUPPRESSED BY AN HP COMMAND.

STATES -BEGINS STATES APPLIED ANALYSIS AND INITIATES THE FAULT SIMULATION PREPROCESSOR. STATES APPLIED ANALYSIS LISTS WHICH FAULTS ARE NOT POTENTIALLY DETECTABLE. THE TEST FOR POTENTIALLY DETECTABLE IS [illegible] A REQUIRED INPUT STATE TO DETECT A FAULT HAS BEEN APPLIED TO A GATE. THE STATES APPLIED [illegible] GIVES A QUICK UPPER BOUND ON THE FAULT COVERAGE OF A GIVEN INPUT SEQUENCE.

TITLE -PRINTS A TITLE AT THE TOP OF EACH PAGE CONSISTING OF THE LITERAL STRING ([illegible] OF 60 CHARACTERS) FOLLOWING THE WORD TITLE.

C3. SIMULATION CONTROL COMMANDS.

THESE COMMANDS CONTROL THE STEPS OF THE SIMULATION.

CALL X -CALLS SALSIM SUBROUTINE X. NO PARAMETER PASSING EXISTS AS ALL VALUES IN SALSIM ARE GLOBAL (LIMIT OF 50 ROUTINES) THERE IS A LIMIT OF 50 SUBROUTINES IN EACH SALSIM PROGRAM.

CONTINUE -THIS IS [illegible] NO OP USED AS A LABELLED STATEMENT TO TERMINATE LOOPS OR AS A TARGET [illegible] A GO TO.

IF (LOG COND) [illegible] -[illegible]
ZZZ IF [illegible].

IF (LOG COND) [illegible] ELSE -[illegible]
TRA[illegible]
THE LOGICAL [illegible]
XB [illegible].

IF? ZZZ -CAUSES [illegible]
IF [illegible].

ITA N -INITIALIZE [illegible] TIME TO THE VALUE N.

RESET [illegible] -THE RE[illegible] STATE [illegible] OF THE SIMULATION
FROM A [illegible]. A [illegible] IN
THE MAIN [illegible] TO [illegible].

STOP -STOPS THE SIMULATION.

SU (LOG COND) -SIMULATES IN THE 4 -STATE MODE UNTIL THE LOGICAL
CONDITION IS TRUE.

SUS (LOG COND) -SIMULATES IN THE 8 -STATE MODE UNTIL THE LOGICAL
CONDITION IS TRUE.

```
        EX. [illegible]

        PC [illegible]
        PRI[illegible] X1,[illegible]
        SEQ [illegible]
        S[illegible] PC [illegible] X2
        SU (T[illegible])
        IF* [illegible] MODES ARE [illegible].
        IF (O[illegible]) CALL S[illegible]
        PC ALL D[illegible].
        STOP
        END
S[illegible]    PC [illegible]
        RESTORE [illegible]
        DO TAC1 [illegible]
        SEQ [illegible] X1,[illegible]
        SU ([illegible])
TAC1    CONTINUE
        RETURN
        END
```

[illegible]

SO[illegible] COMMAND	[illegible] TY[illegible]	[illegible]
CALL	C3	CALLS [illegible]
CONT[illegible]	C3	NO OP [illegible] TO [illegible] A LABEL.
D[illegible]	C3	[illegible] DO [illegible].
DO WHI[illegible]	C3	[illegible] A DO [illegible] A [illegible]
DUMP	C2	WRITES NODE [illegible], TO [illegible].
END	C3	T[illegible] A SA[illegible].
GO TO	C3	TRA[illegible] TO SPECIF[illegible] LABEL.
HAZARD	C2	TURNS ON OR OFF [illegible] ANALYSIS.
HP	C2	HALTS PRINTING.
IF	C3	CONDITI[illegible] EXEC[illegible] OF A S[illegible] STATEMENT.
IFTH[illegible]	C3	CON[illegible] T[illegible] OF EXECUTION CONTROL.
IF?	C3	TEST FO[illegible] ANY CIRCUIT NODE WITH VALUE ?.
IT?	C1	INITIALIZE ALL NODES TO VALUE ?.
ITA	C3	INITIALI[illegible] S[illegible]P.
P[illegible]	C2	PRINT C[illegible].
PCO	C2	PRINT [illegible] VALUES FOR CHANGES ONLY.
PE	C2	PRI[illegible] SO MANY TIME STEPS.
PRINT	C2	PRI[illegible] OF [illegible]S.
PV?	C2	PRI[illegible] WITH VALUE OF ?.
RESTORE	C1	REST[illegible] TO CIRCUIT CO[illegible] A 'SET TO' NODE.
RETURN	C3	RET[illegible] FROM SA[illegible]ROUTINE.
SEQ	C1	SEQ[illegible] VALUES.
SET TO	C1	SETS [illegible] NODE TO A VALUE.
SP	C2	START [illegible] VALUES.
STATES	C2	START OR STOP STA[illegible] ANALYSIS.
STOP	C3	STOP [illegible].
SI[illegible]	C3	SI[illegible] SPE[illegible] CO[illegible].
S[illegible]	C3	SI[illegible] SPE[illegible] CON[illegible].
TITLE	C2	PRINT A TITLE AT [illegible] PAGE [illegible].

1 [illegible]

2 [illegible] (c[illegible]).

3 [illegible] by [illegible].

4 More than [illegible] source line. [illegible] ignored.

1 01 More than [illegible] Further lines ignored.

1 02 Tag on continuation line. Tag ignored.

7 01 Internal flag not 1 or 2. Call to this subroutine is ignored.

7 02 More than 600 variable [illegible] table overflow. Excess over 600 not stored.

8 01 Multiply compound statement. CONTINUE inserted to replace each beyond two.

8 02 Blank line. Ignored.

8 03 Invalid op code. CONTINUE inserted.

8 04 Internal inconsistency. Cannot [illegible] following [illegible]. CONTINUE inserted.

8 05 Unmatched [illegible]. No action taken.

8 06 Unmatched [illegible]'s. No action taken.

8 07 Unrecognizable operand. Should be ON, OFF, or blank. CONTINUE inserted.

8 08 Length of a caption or title exceeds 80 characters. Truncated to 80.

8 09 [illegible] of the form [illegible] has been omitted. N is assumed to be 0, n is assumed to be 1.

8 14 [illegible]

8 15 [illegible]

8 16 [illegible] Con[illegible]

8 17 No right parenthesis [illegible] logical condition. CONTINUE [illegible]

8 19 Excess characters follow logical condition. Ign[illegible]

8 20 Left parenthesis should [illegible] first non-blank following op code. CONTINUE inserted.

8 21 SU [illegible] to SUB [illegible] to preceding EIGHT instruction.

8 22 SUTS [illegible] to SUTS [illegible] to preceding EIGHT instruction.

8 23 S[illegible] to SU [illegible] to preceding FOUR instruction.

8 24 [illegible]

8 25 [illegible]

8 26 [illegible] left [illegible] parenthesis [illegible] inserted.

8 27 [illegible] CONTINUE inserted.

8 28 [illegible] CONTINUE inserted.

[illegible]

8 3[illegible] [illegible]

8 [illegible] [illegible]

8 34 No operand. Logical symbol [illegible].

8 35 Excess characters follow logical symbol. [illegible].

8 36 Logical symbol is invalid. CONTINUE inserted.

8 37 No operand. Statement is replaced by CONTINUE.

8 38 Internal error. Cannot locate leftmost position of operand. CONTINUE inserted.

8 39 Operand not a number, but does not start with *. CONTINUE inserted.

8 40 Cannot locate + that should follow *. Statement replaced by CONTINUE.

8 41 Character following * is not +. CONTINUE inserted.

8 42 Number should follow *+. No number found. Statement replaced with CONTINUE.

8 43 Field following *+ is not numeric. CONTINUE inserted.

8 44 SEQ has no operand. Replaced with CONTINUE.

8 45 No left parenthesis in line. CONTINUE inserted.

8 46 Invalid starting value. Length not 1 or 4. CONTINUE inserted.

8 47 Invalid starting value. One character, but not 0 or 1. CONTINUE inserted.

8 5[illegible] [illegible]

8 52 [illegible]

8 53 [illegible]

8 54 [illegible]

8 55 No operand. CONTINUE inserted.

8 56 No left parenthesis found to delineate logical expression. CONTINUE inserted.

8 57 Mismatched parentheses. Cannot locate logical expression correctly. CONTINUE inserted.

8 58 Logical expression cannot be evaluated. CONTINUE inserted.

8 59 No statement follows logical condition. CONTINUE inserted.

8 60 Word THEN not followed by blank in IF-THEN statement. Not translatable. CONTINUE inserted.

8 61 [illegible] CONTINUE inserted.

8 62 [illegible] CONTINUE inserted.

8 63 Word ELSE not followed by blank in IF-THEN-ELSE statement. Not translatable. CONTINUE inserted.

8 64 No label follows word ELSE. CONTINUE inserted.

8 65 [illegible] is blank. CONTINUE inserted.

8 66 Only one operand. Invalid format. CONTINUE inserted.

8 71 [illegible]

8 72 [illegible]

8 73 [illegible]

8 74 [illegible]

8 75 More than 4 nested [illegible] CONTINUED.

8 76 No operand. [illegible]

8 77 [illegible] left parenthesis [illegible] parenthesis. [illegible]

8 78 (GT) [illegible], but can't find [illegible]

8 79 (GT) [illegible], but [illegible] is not [illegible] COUNT inserted.

8 80 No variable list. [illegible]

8 81 No valid variable name [illegible] inserted.

8 82 No operand. [illegible]

8 83 Operand [illegible]

8 84 No operands. [illegible] inserted.

8 85 First word SET. [illegible] COUNT inserted.

8 86 Only one operand. Should be [illegible]

8 87 Only one operand. Should be 2. [illegible]

8 88 More than [illegible] states listed. [illegible]

[illegible]

18.04 [illegible]

18.05 [illegible] not [illegible]

18.06 Length of [illegible] is not [illegible].

18.07 [illegible]

18.08 [illegible]

[illegible]

24 01 [illegible]

24 01 [illegible]

24 02 [illegible]

24 03 [illegible]

25 01 [illegible]

26 01 [illegible]

26 02 [illegible]

26 03 [illegible]

[illegible]

27 01 [illegible]

27 02 [illegible] inserted.

27 03 [illegible] not found for [illegible]

[illegible] pair of parenthe[illegible]

[illegible]

28 01 [illegible]

28 02 Multiple operators found during scan. CONTINUE inserted.

28 03 Relative time may not be left operand. CONTINUE inserted.

28 04 No right operand found for operator. CONTINUE inserted.

28 05 Multiple operators found during scan. CONTINUE inserted.

28 06 Internal error. Type of operand not an allowed code. CONTINUE inserted.

28 07 Illegal collation of operands. CONTINUE inserted.

28 09 [illegible]

28 10 [illegible]

32 01 Character Table 1. Ignored.

APPENDIX [illegible]

THE [illegible] SOFTWARE

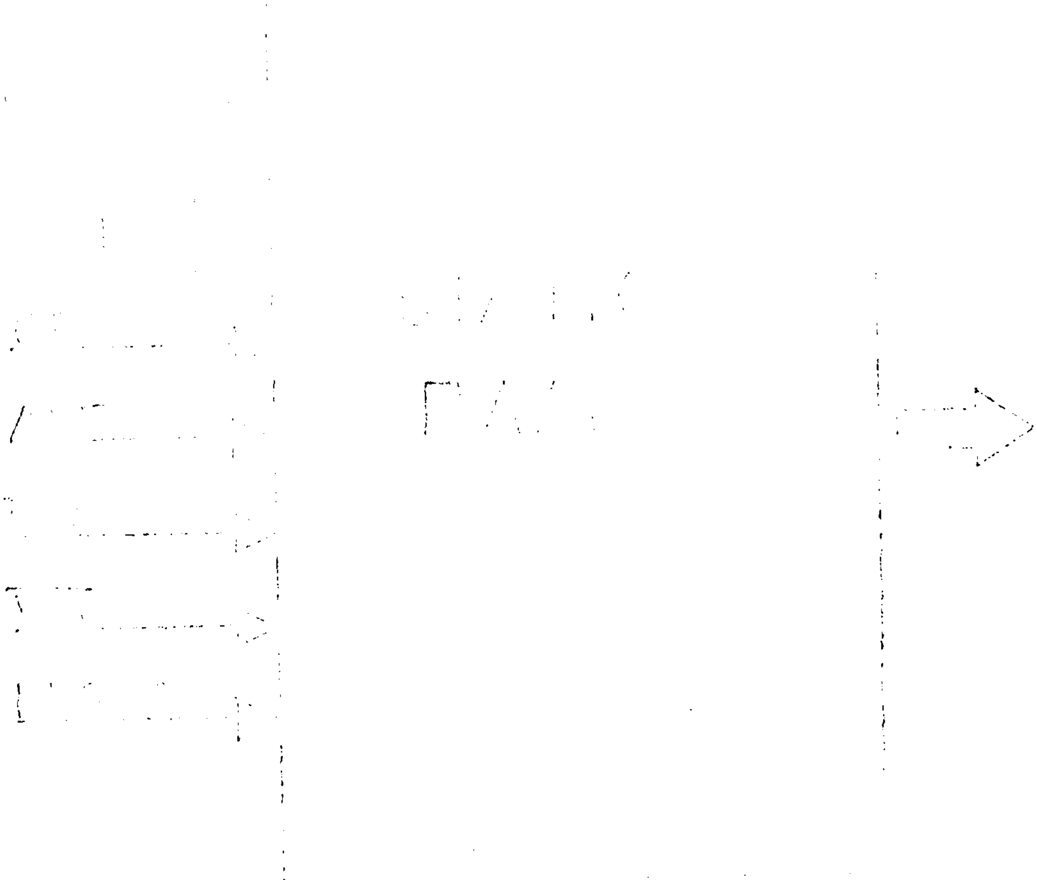
```
      [illegible]
C
C [illegible]
C

      [illegible]
C
      [illegible]
      [illegible]
        [illegible]
        [illegible]
10    [illegible]
      J[illegible] (1)[illegible](2),[illegible](3),
     1 [illegible](5))
      [illegible]
100   J[illegible] (1) 3
      RE[illegible]
      END
C
C
C
      SUB[illegible] OR[illegible]
C
C SETS UP [illegible] O[illegible] A[illegible] FOR [illegible] FOUR IN[illegible] OR GATE
C
      IN[illegible]
      CO[illegible](7,7,7,7),IFLAG,I[illegible](8)
      D[illegible]/3,[illegible]/
C
      IFLA[illegible] 1
      OPE[illegible]'[illegible]',MODE='ASCII',
     1 DIS[illegible]='SA[illegible]',FI[illegible]'[illegible]RY')
      DO 200 I=1,7
      DO 200 J=1,7
      DO [illegible] K=1,7
      DO 200 L=1,7
        READ (6[illegible],1[illegible]) [illegible](I,J,K,L)
100     F[illegible] (I1)
200   CO[illegible]
      CL[illegible]=60)
      RET[illegible]
      E[illegible]
C
C
C
```

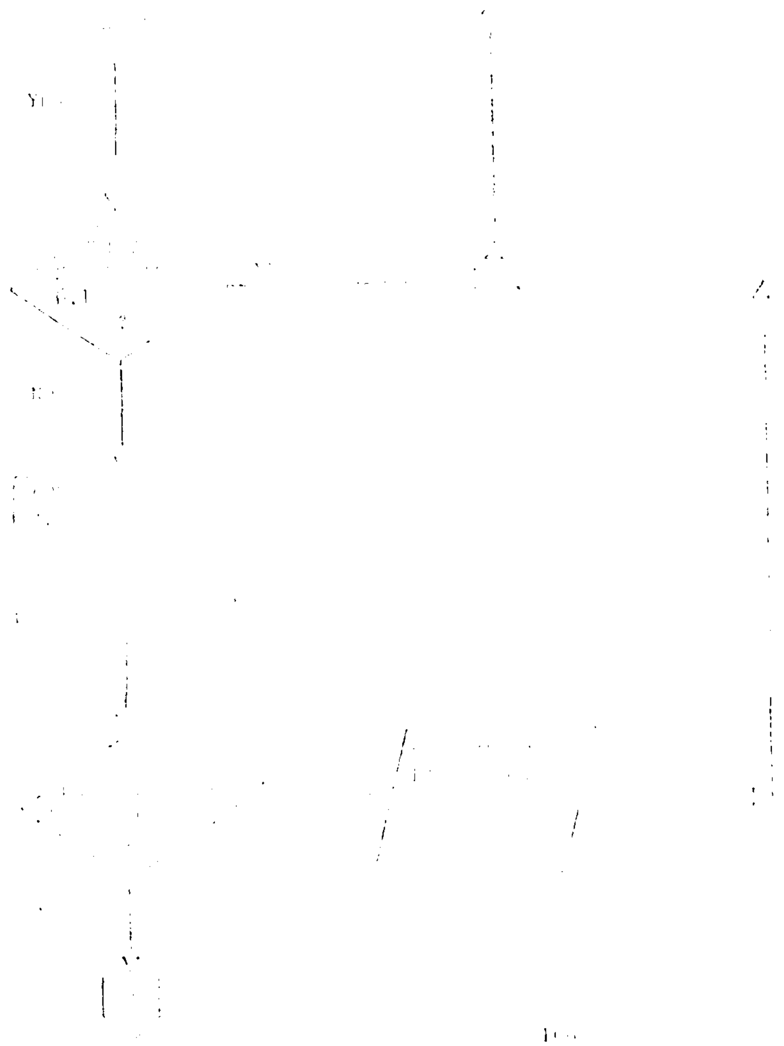
```
      [illegible]
C
C
C [illegible]
C [illegible]
C
C [illegible]
C [illegible]
C.....
C [illegible]
C
      [illegible]
      [illegible]
      [illegible]
      [illegible]
C
      [illegible]
10[illegible] [illegible]
      [illegible]
      [illegible]
      [illegible]
      [illegible]
      JOUT(1) [illegible](1),[illegible](2),[illegible],INPUT(4))
      JOUT([illegible]) [illegible](2),[illegible]
      JOUT([illegible]) [illegible]
      JOU[illegible] [illegible](1),[illegible]
      JOUT(5) [illegible](1),[illegible]
      JOU[illegible] [illegible](1),[illegible]
      JOUT([illegible]) [illegible](2),[illegible]
      JOU[illegible] [illegible]HOLD(4))
      JOUT([illegible]) [illegible](2),[illegible]
      JOUT(10)=[illegible]IHOLD(4))
      JOUT(11) [illegible]
      JOUT(1[illegible]) [illegible](1),[illegible](2),[illegible](3),[illegible]HOLD(4))
      JOUT([illegible]) [illegible](1),[illegible]
      JOU[illegible] [illegible]HOLD([illegible]))
      JOU[illegible] [illegible](2),[illegible]
      J[illegible] [illegible]
      RE[illegible]
      END
C
C
C
```

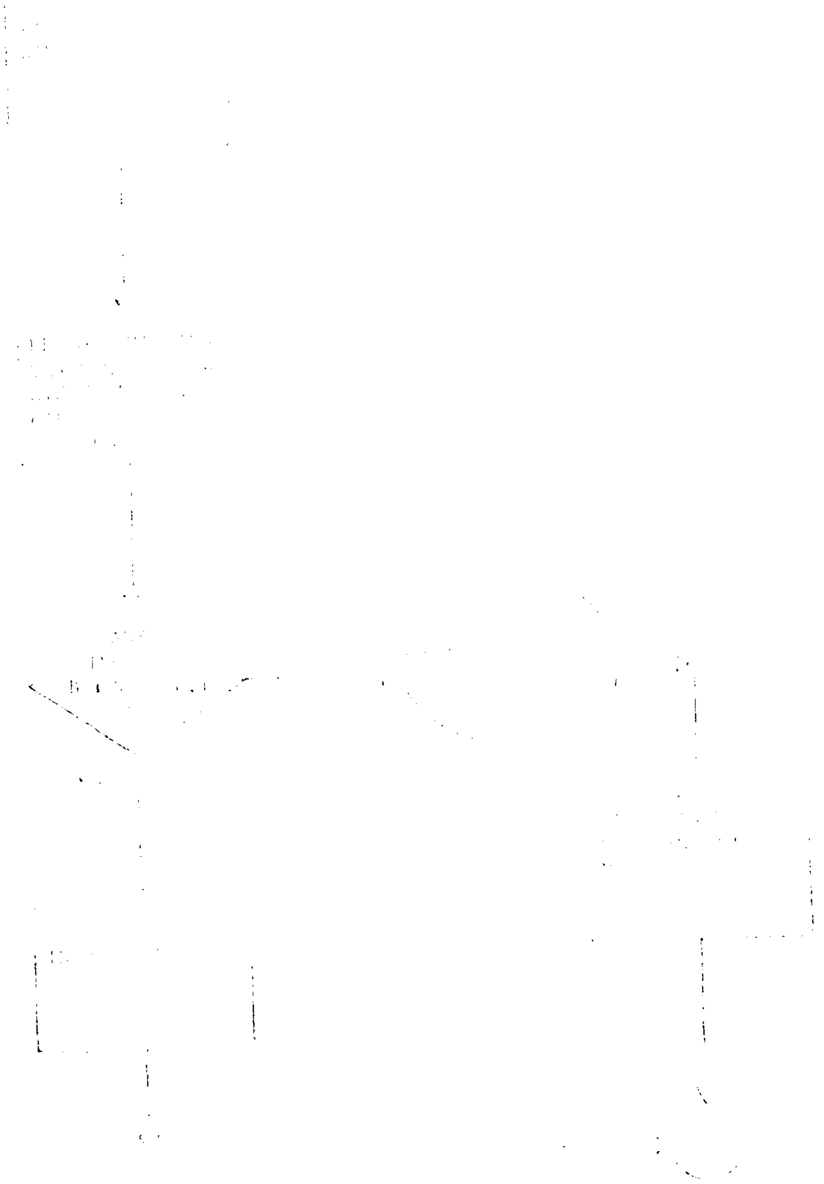
```
      F[illegible]ORSET[illegible](L[illegible],[illegible],[illegible]RE3,[illegible]4)
C
C [illegible]
C
      [illegible]
      D[illegible]
C
      [illegible]
      IF ( [illegible] ) [illegible]
     1 ( [illegible] )
      I[illegible]
      I[illegible]
      IR[illegible]
      IR[illegible]
      IF ( [illegible] ) [illegible](1) [illegible]
      IF ( [illegible].LT.[illegible] ) [illegible](2) [illegible]
      IF ( [illegible].LT.[illegible] ) [illegible](3) [illegible]
      IF ( [illegible].LT.[illegible] ) [illegible](4) [illegible]
      ORS[illegible]([illegible](1),IR[illegible](2),IR[illegible](3),
     1 I[illegible](4))
      RETURN
  100 ORS[illegible]=3
      RETURN
      END
C
C
C
```

APPENDIX F

THE RAW DATA







```
C
C
C
C [illegible]
C                   [illegible]     - [illegible]
C                   [illegible]     -        2
C                   [illegible]     -        3
C                   AL[illegible]   -        4
C                   [illegible]     -        5
C                   [illegible]     -        6
C                   AD[illegible]   -        7-14
C
C                   DATA    - [illegible]
C
      CO[illegible](40)
      D[illegible]
      DI[illegible]
      I[illegible],OP[illegible]
      I[illegible]
      C[illegible],
     1              [illegible]
      D[illegible],0,0,0/
      DA[illegible],0./
      D[illegible]/
      DATA [illegible]/
      DATA [illegible]/
      DATA [illegible]/
```

```
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C          T[illegible]
C [illegible]
C [illegible]
C
```

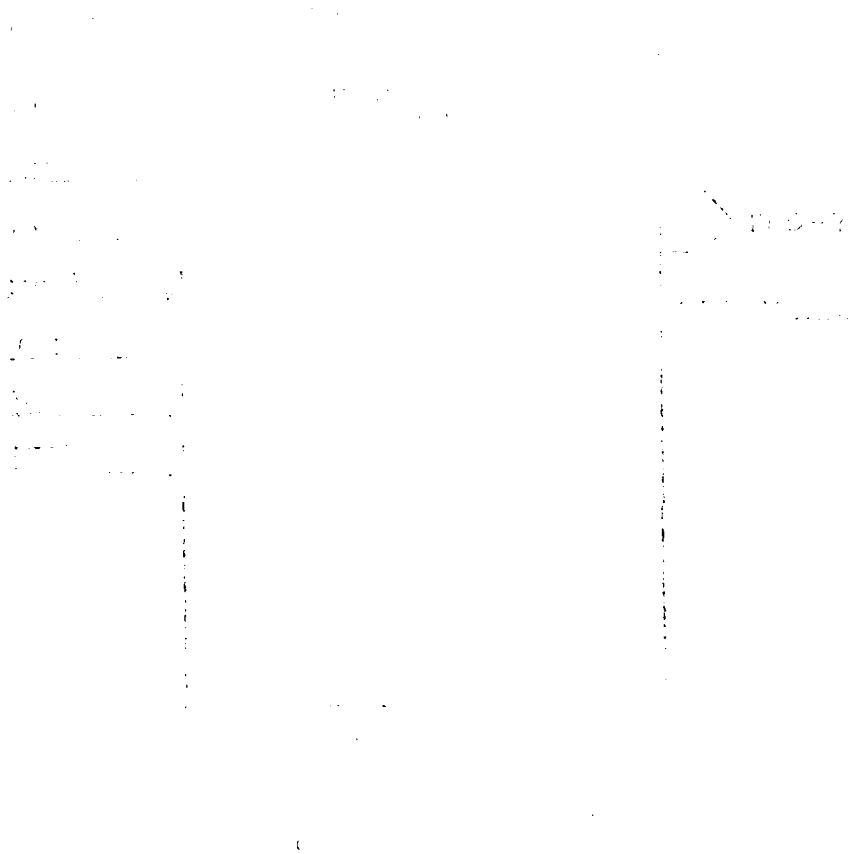
```
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C
5     IF (I [illegible]
     1   [illegible]
      DO [illegible]
        J [illegible]
        IF (I [illegible]
     1      IF [illegible] (J)
        IF (G [illegible]
        IF (C [illegible]
        IF (G [illegible]
10      G [illegible]
        IF( [illegible]
     1      I [illegible]
        IF( [illegible]
     1      I [illegible]
      DO [illegible]
        IF [illegible]
        IF [illegible]
12    C [illegible]
      IF (C [illegible]
     2   [illegible]
     3   G [illegible]
      [illegible]
     1   [illegible]
      [illegible]
     1   [illegible]
      IF (C [illegible]
     1   [illegible]
     2   [illegible]
15    [illegible]
     1   [illegible]
     2   [illegible]
     2   [illegible]
     3   [illegible]
      [illegible]
      [illegible]
```

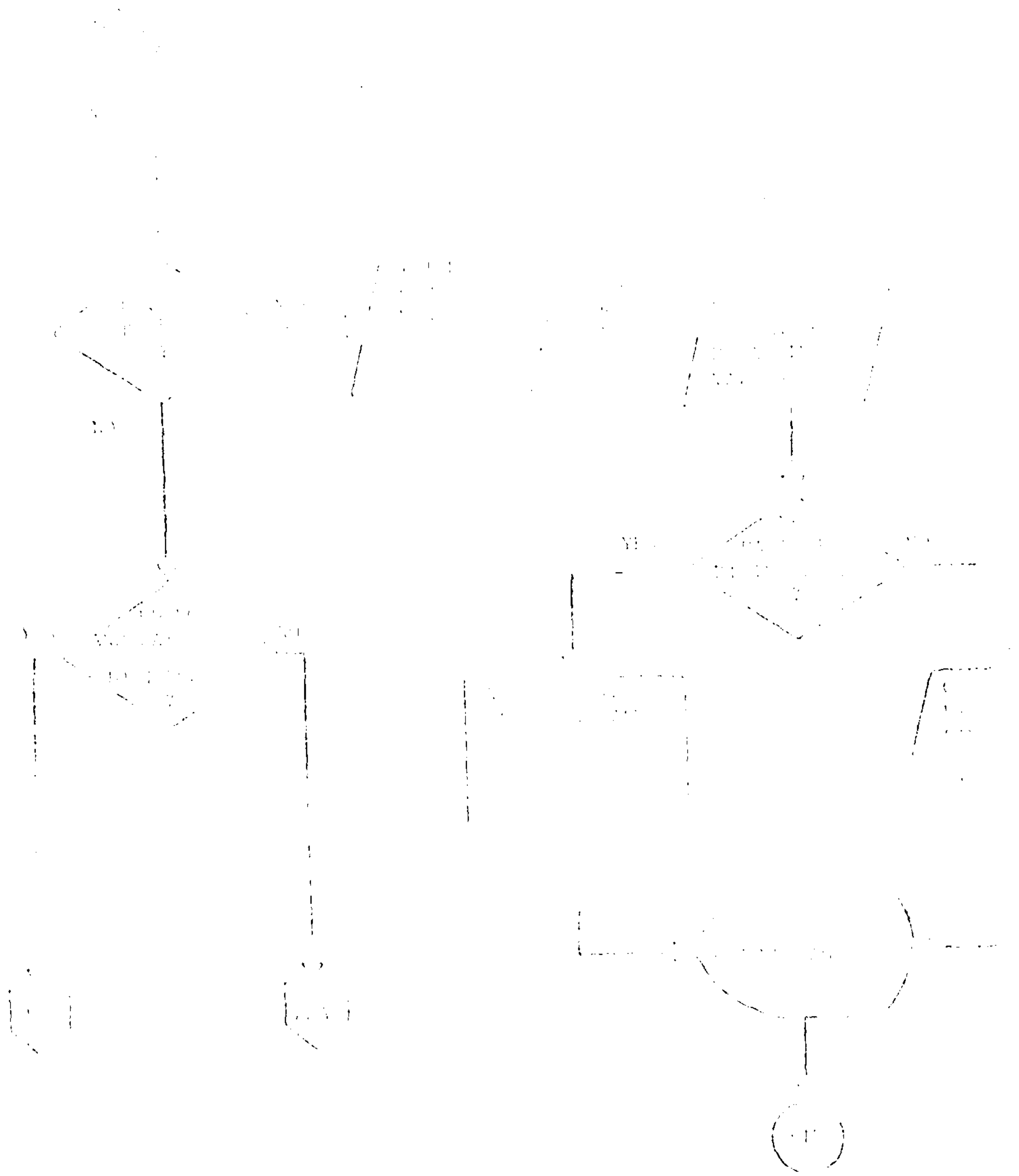
```
C
C
C
C
C
C
C
C [illegible]
C [illegible]
C
      IF (([illegible]).[illegible]) GO TO [illegible]
C
C [illegible]
C [illegible]
C [illegible]
C
      ACC[illegible]=0
      DO 20 [illegible] I=[illegible]
        J=SB[illegible]
        IF ([illegible](I).[illegible].1) GO TO 70
        AC[illegible]([illegible](I,[illegible]))
20    CONTINUE
      IF (ACC[illegible]) GO TO 27
      WRITE ([illegible])
      WRITE ([illegible] A[illegible])
      IF ([illegible]) [illegible] (I,[illegible]) [illegible]
25    FORMAT (/,' [illegible]',
     1  ' AR AR[illegible]',/,' [illegible]',I9,' [illegible]',I9,/)
      LACC=1.
      GO TO 60
C
C [illegible] TO [illegible]
C
27    IF ([illegible]) [illegible] 35
C
C [illegible]
C
      IWO[illegible]
      DO 30 [illegible]
        J[illegible]([illegible], XN[illegible])
        LX[illegible]
30    CONTINUE
      IF ([illegible].0) GO TO 101
      [illegible]
      [illegible]
      IF ([illegible])
3[illegible]  FORMAT (' [illegible])',/,
     1  ' [illegible]',I10,/)
      [illegible]
      GO TO 7[illegible]
```

```
      [illegible]
57    [illegible]
     [illegible] (1X,/)
      [illegible]
      GO TO 100
C
C [illegible]
C
60    DO 65 I=2, [illegible]
         JO[illegible] [illegible]
65    CONTINUE
      [illegible]
      [illegible]
      IF ([illegible]) (1,67)
67    [illegible] NO GENE[illegible] DATA LINES',/)
      [illegible]
      [illegible]
C
C [illegible] TO [illegible] LINE
C
70    DO 75 [illegible]
         JO[illegible]
75    [illegible]
      [illegible]
C
C [illegible]
C
100      DO 110 [illegible]
            [illegible]
110      CONTINUE
      RETURN
      END
C
C
C
```

APPENDIX G

THE [illegible]





```
C
C
C                   [illegible]     = [illegible] (1)
C                   [illegible]     =        [illegible]
C                   [illegible]     =        [illegible]
C                   [illegible]     =        4
C                   [illegible]     =        5
C                   [illegible]     =        6
C                   [illegible]     =        7
C                   [illegible]     =        8-18
C
C                   P[illegible]    = [illegible] (1)
C                   D[illegible]    =        2-9
C
      COM[illegible]/[illegible](40)
      D[illegible]
      D[illegible])
      I[illegible],OF[illegible])
      I[illegible]
      CO[illegible]S/[illegible],ZZ[illegible]ZZ1,
     1         12[illegible])
      DATA [illegible]/[illegible],0,1,0,2,0,0/
      DATA [illegible]/[illegible]
      DATA L[illegible]/,[illegible]7777/
      DA[illegible]
      DATA RO[illegible]1,1/
      DA[illegible],1/
      DA[illegible]/
```

```
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C [illegible]
C
```

```
      IF ( [illegible] ))
     1     GO TO [illegible]
      [illegible]
        [illegible]
        IF ( [illegible]
     1     [illegible]
        IF ( [illegible] )
        IF ( [illegible]
        IF ( [illegible]
10    CONTINUE
        IF( [illegible]
     1     [illegible] (3)
        IF( [illegible]
     1     [illegible]
        IF( [illegible]
     1     [illegible] (3)
      DO [illegible]
        IF ( [illegible]
        IF ( [illegible]
[illegible]  [illegible]
      IF ( [illegible]
     1     [illegible]
     2     [illegible]
     3     GO TO [illegible]
      WRITE ( [illegible] ).
     1     [illegible]
      [illegible] ( [illegible] ),
     1     [illegible]
      [illegible]
     1     [illegible]
     2     [illegible]
[illegible]  [illegible]',
     1     [illegible]
     2     [illegible]
     3     [illegible]
     4     [illegible] ')
      [illegible]
```

```
C
C [illegible]
C [illegible]
C
      IF (G[illegible](I).[illegible]0[illegible]
C
C [illegible]
C [illegible]
C IF [illegible]
C
      IF ([illegible]0.) [illegible](I) [illegible]
       [illegible]
       DO [illegible]
         J[illegible]
         IF ([illegible]
         A[illegible]
20     [illegible]
      IF (A[illegible]
      [illegible]
      W[illegible]
      IF ([illegible]
25    FORMAT ([illegible]
     1 [illegible]
      [illegible]
      GO [illegible]
C
C [illegible]
C
[illegible]
```

```
C
C
C

67      ...
        RETURN
        END
C
C
C
70      DO 75 ...
           ...(...)=1
75      CONTINUE
        GO TO 100
C
C
C
100        ...
              ...
110        CONTINUE
           RETURN
           END
C
C
C
           SUBROUTINE ...
C
C
C
           COMMON ...
      1
C
           DO 10 ...
10         CONTINUE
20
```

AD-A100 784 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2

A FUNCTIONAL LEVEL PREPROCESSOR FOR COMPUTER AIDED DIGITAL DESI--ETC(U)

DEC 80 P G RAETH

UNCLASSIFIED AFIT/GCS/EE/80D-12 NL

3 OF 3
AD
A100 784

END
DATE
FILMED
7 81
DTIC

APPENDIX B

RUNNING SALOGS/SISL

```
TO [illegible] THE SISL [illegible]:

EX SISL


[illegible]:

.run dlink
/ots:[illegible]/seg[illegible]nt:low -
ACKP[illegible],[illegible],NETW[illegible]
N[illegible]WRK/[illegible]/G
RUN NETW[illegible]


TO COMPILE A SALOGS SYSTEM EXERCISING PROGRAM:

.run dlink
/ots:nonshar/segment:LOW/COMMON::50000 -
SALSIM
SALSIM/SAVE/G
RUN SALSIM


TO RUN THE TWO PREVIOUSLY COMPILED PROGRAMS:

LOAD %"COMMON::100000" SIMUL,FUNCNS,LATEST,[illegible]10,[illegible],ACKSUB/LIB,ACKEN
SAVE SIMUL
RUN SIMUL
```


THE SEVERAL NAMES USED HERE ARE USER DEFINED PROGRAM NAMES EXCEPT:

LOAD, COMMON, OTS, NONSHAR, SEGMENT, LOW, DLINK, SAVE, RUN, EX, G

DETAILS OF THEIR MEANING MAY BE DERIVED BY CONSULTING THE CURRENT

DEC SYSTEM 10 OPERATING SYSTEMS MANUAL.

GLOSSARY

GLOSSARY

[illegible] [illegible] rather than [illegible] [illegible]. The designer is more interested [illegible] that [illegible] have occurred rather than what the chip will really do.

Black box- a functional level of modeling which ignores the future contents of a given module. This module simply delivers as output the "stored" word or the default word.

Core- a computers' main memory. This main [illegible] include swap-out disk storage space and memory from which instructions are directly executed.

CPU Time- time spent by the central processor during the execution of a given job.

Functional level modeling- specifies the connections between behavioral models. It will combine the capabilities of several behavioral models. This level employs structural and behavioral modeling.

Gate level modeling- modeling using the basic logical primitives such as AND, OR, NOR, etc. This level is more concerned with the actual operation of a chip given undefined inputs.

Intermix- to use two or more ideas, or simulations at the same time.

Library- a series of computer subprograms accessible by any number of calling programs. These perform certain operations wich are common to the calling programs.

Link- to connect to. When a computer program is loaded into memory, it can consist of code the parts of which have been written independently of it.

Logical link- Program subroutines can be called in any given order. One must define that order as well as make the subroutines available to the calling program.

Macro- perhaps the most confusing word. As a level of modeling, it refers to the big picture of the interconnections and performance of large subsystems. As a piece of software, it is a block of code which the compiler or assembler can place at any of several specified locations of a progam.

Parsing- the procedure by which the phrases in a string of characters are associated with the component names of the language grammar which generated the string [Ko,1984].

Register transfer modeling- [illegible] the operation of a digital system by sp[illegible]ying [illegible] and when data is passed.

Real Time System- A computer [illegible] in [illegible] different forms. [illegible] by the central processor.

Structural level modeling- defines the interconnections of signal paths.

SET TO Words- A SALOGS node may be given a standard, never to change value by the designer. This value will remain regardless of the simulation produced value.

Subsystem- a part of a complete digital unit. For example: a ROM is a subsystem to a computer memory.

Wall Time- the total time the computer has control of a given program. This time spans the moment a job first enters the execute queue and the moment it enters the final output queue.

VITA

Peter G. Raeth was born on 10 July 1951 in Jackson Michigan, the son of Nicholas Conrad Raeth and Theresia Roehm Raeth. In 1970 he enlisted in the United States Air Force. He was Honorably Discharged in 1976. In 1975 he graduated from the Trident Technical College at Hanahan South Carolina with an Associate in Electronics Engineering Technology. That same year he began the four year engineering program taught by the University of South Carolina at Columbia. During the period 1975-1979 he studied digital engineering, attended USAF-ROTC, and worked as a free-lance consultant in software applications, twice publishing his research. In 1979 he graduated with a Bachelor of Science in Electrical Engineering and was commissioned a Second Lieutenant USAFR. He is a member of Tau Beta Pi, Eta Kappa Nu, and Omicron Delta Kappa. His first assignment was to attend, in residence, the Masters program in Computer Engineering given by the Department of Electrical Engineering of the Air Force Institute of Technology at Wright-Patterson Air Force Base Ohio.

Permanent Mailing Address:

2Lt Peter G. Raeth

5752 Chatham Avenue

Hanahan, S.C., 29406

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/80D-12	2. GOVT ACCESSION NO. AD-A100 784	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A FUNCTIONAL LEVEL PREPROCESSOR FOR COMPUTER AIDED DIGITAL DESIGN		5. TYPE OF REPORT & PERIOD COVERED MS THESIS
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) PETER G. RAETH		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AIR FORCE INSTITUTE OF TECHNOLOGY (AFIT-EN) WRIGHT-PATTERSON AFB OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS AIR FORCE INSTITUTE OF TECHNOLOGY (AFIT-EN) WRIGHT-PATTERSON AFB OH 45433		12. REPORT DATE DECEMBER 1980
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release;
IAW AFR 190-17
Fredric C. Lynch
Fredric C. Lynch, Major USAF
Director of Public Affairs

1 JUN 1981

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

MODELING	COMPUTER-AIDED DESIGN	RAM
SIMULATION	SALOGS	ROM
DIGITAL	SISL	MODELING LANGUAGES
BEHAVIOR MODELING	GATE LEVEL MODELING	DIGITAL SYSTEMS-SIMULATION
FUNCTIONAL MODELING	DECODERS	SIMULATION LANGUAGES

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

SEE PAGE 191

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

[illegible]

[illegible] cannot be [illegible] in gate level and [illegible] level. [illegible] is to create a functional level preprocessor [illegible] functional device models linked to a gate level simulator's input language. This permits the mixing of behavioral models with gate level models in the same system structure. The combination of processes (element models or primitives) and their structure (interconnections) can be exercised all at one time during a single simulation session. From the start, there came forth an obvious method which could be used to intermix the several levels of modeling.

Two separate pieces of software were written to implement a specific solution to the above stated situation. SISL, Structural Interface to the Salogs Language was created. This is a functional level preprocessor to SALOGS (SAndia LOGic Simulator) which is an eight-state, MOS, gate level digital systems simulator. SISL will accept functional level system descriptions and convert them to a form acceptable to SALOGS.

The other effort was the building of a functional level modeling library. This library consists of three behavior models: a 4-16 decoder, a 2048 x 8 ROM, a 256 x 8 RAM. These models are designed to be used in a functional level/gate level model of a digital system and will link to the SALOGS run time system. Together, these two programs (SISL and the modeling library) provide the easy use of the top-down approach to digital system design.

[illegible]

[illegible] CLASSIFICATION

DATE
FILMED
8